A FRAMEWORK FOR THE ANALYSIS OF SOPHISTICATED CONTROL

Robert C. Whitehair

UMass CMPSCI Technical Report 95-******
February 1996

Department of Computer Science
University of Massachusetts
Amherst MA 01003-4610

EMAIL: *whitehair@cs.umass.edu*

This work was supported by DARPA contract N00014-92-J-1698, Office of Naval Research contract N00014-92-J-1450, and NSF contract CDA 8922572. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

A FRAMEWORK FOR THE ANALYSIS OF SOPHISTICATED CONTROL

A Dissertation Presented

by

ROBERT C. WHITEHAIR

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial
fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 1996

Department of Computer Science

To my parents, Charles and Margaret Whitehair.

ACKNOWLEDGMENTS

My father was a scientist. When I was growing up, he used to make me read biographies of famous scientists. He used to tell me, "You never know what they are going to ask at your thesis defense." He told me this when I was in sixth grade.

Through his assigned readings and many hands on learning experiences I won't describe here (they involve the word "eviscerate"), my father impressed on me the importance of science as the study of the natural world. In essence, he taught me that science is a process of building models that describe the natural world, testing the accuracy of the models, and then building new models based on the results of the testing. This impression has had a profound influence on me and the work described in this thesis. As a direct consequence of it, I have sought to develop a framework that supports the study of the aspects of computer science and artificial intelligence that can be said to exist in the natural world. (Sorry, if you want more details you will have to read the thesis. If you are hesitant, I am happy to encourage you by stating that the word "eviscerate" does not appear again in this dissertation.) Thanks, Dad. This is for you.

I want to thank Victor Lesser for his part in creating this framework. His most important contribution may not be recognized by those who actually read this thesis. This contribution is the wealth of ideas, many of them profound, that Vic has amassed during his career. This thesis was motivated by a desire to better understand a small portion of these ideas and their relationship to other profound ideas and to models of the natural world. I also want to thank Victor for all his support and friendship during our years together.

Michele Roberts – Thanks! Without you, I have no doubt that Vic's research groups would have self-destructed long ago!

I want to thank everyone on my committee for their input and suggestions. Norm Carver is clearly owed the largest allotment of gratitude. His comments on previous papers, suggestions, questions, and guidance were extremely valuable. In addition, Norm's work on differential diagnosis is one of the significant concepts that my framework is intended to study. Also, Shlomo Zilberstein, Allen Hanson, and George Avrunin made important suggestions and comments that have been incorporated into this dissertation and I would like to thank them for their efforts.

I cannot imagine a more talented, creative, intelligent, and enjoyable environment in which to study than what I experienced at Umass. For this I would like to thank all the members of the Distributed AI Lab and the Umass community at large. This includes (in no particular order) Dan Neiman, Keith Decker, Allen "Bart" Garvey, Zarko Cvetanovic, Dave Westbrook, Teri Westbrook, Dave Hildum, Marty Humphrey, Ed Durfee, Joe Hernandez, Frank Klassner, Dorothy Mammen, Malini Bhandaru and many others. I especially want to thank Maram "Naghi" Nagendraprasad and Toumas Sandholm for their comments and contributions. Toumas, we'll get that chess game in yet! (Sorry, but if I mention Dave Hart, then I have to mention Al Kaplan, Lori Molesky, Gary Wallace, Bob Cook, and Zack Rubinstein, and I am never sure how to spell "Rubinstein" so I will have to thank all of you en masse along with Steve Bradtke and Tony Hosking. Sorry I can't fit you all in!)

Gary and Melanie Tallmon, thank you. I am not sure what you contributed to this thesis, but you are great friends. (Gary told me he would like to see his name in a book someday. This is the least I can do for him.)

I guess I better thank Jim Sleight while I am mentioning friends. I don't know what I would have done without his insightful email and wise council. Jim, this is for you: "xxoo." (Anthony, I would thank you, too, but you were off frolicking in Africa.)

Softball and basketball teams: "thanks!" If anything, you delayed the completion of this thesis, but you made it fun!

I feel I must thank many members of the Umass faculty and other academic and industrial researchers. I don't dare try to name names - the list is far too long. I must, however, make one exception – I would like to thank Ludwig "language precedes thought" Wittgenstein. (He is one of my intellectual heros.)

I guess I need to thank John Forsyth for an untold number of things, including telling me to go to Umass and giving me an instructor's job at MSU. When I think about it, if it hadn't been for John luring me out of a lucrative, high-paying industrial job. Without his wisdom and guidance, I never would have chosen this path. Thanks for saving me, John! I would buy you lunch but – well, you probably remember what it is like to be a poor grad student.

I also want to thank Glen Keeney. His persistent cynicism and goading are what really pulled me through this ordeal! (You're it!) Glen, I WILL buy YOU lunch.

Thank you to everyone who contributed to the efforts to apply these ideas in the "real-world." Thanks Ian, Pete, Monty, Vladimir, Igor, and everyone!

Thank you, family, I love all of you. Thanks for all the support, Stan. I know you tried not to ask, "Are you done yet?" I really appreciate it. Mom, you're the greatest. Thanks for the cookies and for not making me dissect anything. Thanks, Ethel, wherever you are! You are a real inspiration!

Finally, thanks Kathy. For everything. ("More than *anything!*")

ABSTRACT

A FRAMEWORK FOR THE ANALYSIS OF SOPHISTICATED CONTROL

FEBRUARY 1996

ROBERT C. WHITEHAIR

B.S., MICHIGAN STATE UNIVERSITY

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Victor R. Lesser

This dissertation addresses problems associated with the lack of design theories for AI problem solving systems. The principle focus of the work is the introduction and demonstration of a framework for the analysis of sophisticated search control architectures applied in complex problem domains. The thesis associated with this work is that real-world problem domains and problem solving architectures can be represented formally and that these representations can be used to analytically predict and explain a problem solver's performance. Further, the implications of this work are that useful approximations and abstractions can be derived from such formal representations and used to design sophisticated control mechanisms. The ultimate objective of this work is to use these representations as the basis of design theories for building problem solving architectures and dynamic control algorithms.

The framework is based on two formalisms, the *Interpretation Decision Problem (IDP)*, which models both the structure of a problem domain and the structure of a problem solving architecture, and the *UPC* formalism, which provides a general quantitative model of search spaces that can be used in the analysis of problem solving control. Using these models, the problem structures of disparate domains and the problem solving architectures constructed to exploit these structures can be viewed from a unified perspective where control and problem

solving actions can be considered a single class of problem solving activity. Models built from this unified perspective offer advantages for describing, predicting and explaining the behavior of blackboard-based *interpretation* systems and for generalizing a specific problem solving architecture to other domains. Use of the IDP and *UPC* formalisms also supports the synthesis of new, more flexible problem solving architectures.

This dissertation demonstrates how the framework can be applied by analyzing a vehicle monitoring interpretation problem domain and associated problem solving architectures, including a heuristic, multi-level blackboard-based system. Definitions, examples, and experimental results are given for general structures from interpretation problem domains and blackboard-based problem solving architectures. Design principles for general problem solving strategies that exploit the structures are discussed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Though blackboard systems have become one of the standard paradigms used by AI practitioners for building sophisticated knowledge-based systems, there are still no formalisms available for understanding their performance in quantifiable terms. Consequently, at this point in time, the design of sophisticated blackboard-based problem solvers is largely a trial and error process [Carver and Lesser, 1991]. For any given problem solving technique or strategy, there is no formal representation or theory that answers the critical design issues related to *why* it works, *how* it should be applied, *what* domains it can be applied to, and *when* to apply it in those domains. Because of this, the design of blackboard systems, especially their sophisticated control mechanisms, is more of an art than a science and this has lead to a number of significant problems. (These control mechanisms will be referred to as *meta-level operators* or simply *meta-operators* because they reason about issues such as which operator to apply next, what operators can be deleted, how to configure operators, and so forth.) One of the more serious is the lack of predictability. In many cases, until you actually build a system, there is no way to tell how it will perform. Furthermore, the lack of formal specification and representation makes it easy to overlook previous results that might be relevant to the problem at hand. Inefficient approaches might be used when more effective strategies are available. Similarly, if novel solutions are developed, it may not be clear how they are related to other strategies and techniques and it might be difficult to categorize them in such a way that they can be of use to others.

This does not have to be the case. This thesis demonstrates that answers to the critical design issues can, at least in part, be related to specific problem domain and problem solver characteristics that are represented in formal, quantifiable ways. The approach used involves developing formal specifications of the characteristics and inherent properties, or *structure*, of problem solvers and the domains in which they are applied and then using this representation to build a quantitative analysis framework for predicting and explaining the performance of a sophisticated problem solver operating in a complex domain, the signal interpretation problem domain. (Intuitively, interpretation problems are tasks where a stream of input data is analyzed and an explanation is postulated as to what domain events occurred to generate the signal data – the problem solver is attempting to interpret the signal data and determine what caused it [Corkill and Lesser, 1981, Erman *et al.*, 1980].) The specific contributions made by this work include the following.

1. It establishes definitions for a broad class of problems, *sophisticated control* problems, a specific set of problems within this class, the *Interpretation Decision Problem (IDP)*, and a particular type of domain in which these problems occur, *complex domains.*

2. It defines and demonstrates a methodology for formally specifying the structure of a problem domain.

3. It defines and demonstrates a methodology for formally specifying the structure of a problem solving architecture.

4. It defines and experimentally verifies a set of quantitative analysis tools based on the formal definitions of domain and problem solver structure.

5. It defines a formal representation of search spaces and the control of search-based problem solvers.

6. It defines a unifying representation of problem solving that integrates meta-level control processing and base-level problem solving in such a way that their costs and benefits can be directly compared.

7. It introduces and defines an important control concept, *potential*, that is used in the analysis of the long-term costs and benefits of an action.

8. It establishes a set of tools and a framework for experimental analysis and investigation of problem domains and problem solving architectures.

9. It formally defines a set of important sophisticated control mechanisms including preconditions, pruning operators, goal directed processing, and approximate processing.

10. It demonstrates that formal analytical techniques can be applied to heuristic problem solvers that use sophisticated control mechanisms.

11. It defines and demonstrates several prototype design methodologies for constructing sophisticated problem solvers.

12. It demonstrates that formal analytical techniques can be applied to heuristic, approximate processing problem solvers.

13. It defines a general architecture for approximate processing.

To achieve these results, previous work on formalizing theoretical search [Berliner, 1979, Kumar and Kanal, 1988, Pearl, 1984, Stockman, 1979] was extended to model meta-level processing techniques used in blackboard systems. The formalism that is developed is composed of the *Interpretation Decision Problem (IDP)* and the *UPC* formalisms (from *U*tility, *P*robability, and *C*ost) and will be referred to as the IDP/*UPC* framework. The IDP/*UPC* framework assumes that the properties and characteristics of problem domains are structured and that problem solvers can exploit domain structures to more effectively control and focus their search. By explicitly representing the characteristics and properties of a domain and the interrelationships between potential problem solving actions, IDP/*UPC* based tools can be used to predict and explain problem solver performance, design more effective problem solving architectures and dynamic control strategies, and formalize seemingly distinct problem solving strategies in a unified perspective.

The IDP and *UPC* formalisms are each designed to represent different aspects of a domain's or a problem solver's structure. Like Kanal and Kumar's work on formalizing monotonic search algorithms such as branch and bound, A*, B*, and others used in operations research applications [Berliner, 1979, Kumar and Kanal, 1988, Pearl, 1984, Stockman, 1979], the IDP/*UPC* framework specifies the structure of a domain in terms of formal grammars. This

thesis extends previous work in that it eliminates many of the restrictions that were placed on the grammars used, such as monotonicity constraints, and it enhances the grammars used to increase their expressive power. In addition, this thesis extends the representation to include not just the basic search algorithm, but also the problem domain in which a problem solver is deployed and the meta-level processes that constitute a sophisticated control mechanism. These extensions are discussed in more detail in Chapter 2. The IDP formalism uses a context-free grammar and functions associated with production rules of the grammar to represent feature structures in a problem domain and the structure of a specific problem solver. An *IDP generational grammar*, IDP_G, explicitly represents the causes of domain phenomena in interpretation domains such as noise, missing data, and masking. An *IDP interpretation grammar*, IDP_I, represents relationships between problem solving actions, such as "cooperating" or "independent," and the assumptions (implicit and explicit) that meta-operators (i.e., operators that reason about other operators or the general state of the problem solver) make about a domain. The *UPC* formalism represents search space structures in terms of statistical properties derived from a formal specification of a problem domain's structure, such as that provided by an IDP specification. Using the *UPC* representation, we can construct problem solving systems capable of achieving the levels of performance predicted by quantitative analysis of IDP domain specifications and the optimal interpretation control strategy. As a consequence, domains with different structures can be compared using identical evaluation function based control architectures or these architectures can be varied to compare performance of different problem solvers within a given domain.

Figure 1.1 shows a general overview of the analysis framework. As shown in the figure, the natural structure of an interpretation problem is mapped into a formal representation, or *domain theory*, based on the IDP formalism. By representing a domain structure in terms of the IDP formalism, certain statistical information about the structure of the corresponding search space can be derived. In particular, significant relationships among subproblems and among subproblems and final solutions can be determined and, to an extent, quantified. For example, two subproblems might be related in that they both are part of a single solution 40% of the time, or they are both always part of independent solutions, or they are part of competing solutions that are mutually exclusive, etc. Given an IDP specification of an interpretation domain, general characteristics associated with the complexity of problem instances from that domain can be determined. This includes characteristics such as the expected cost of problem solving for a random problem instance. To complement this, the *UPC* formalism explicitly represents subproblem relationships in a quantified way that can be used, either directly or in abstract form, by a problem solver's control component to schedule the execution of problem solving actions. (In a real system, the cost of accurately quantifying relationships may be prohibitively expensive, in which case it is necessary to use approximations or abstractions.) Thus, the *UPC* formalism can be used to determine which actions available to a problem solver are "optimal" from a local problem solving perspective and to explain why a problem solver chose a certain course of action in a given situation.

The combined IDP/*UPC* framework is an initial step needed to formalize complex search processes, such as those associated with blackboard systems [Carver and Lesser, 1991, Corkill, 1983, Erman *et al.*, 1980, Hayes-Roth and Lesser, 1977, Lesser *et al.*, 1989b], that are used in sophisticated interpretation tasks. This approach is critical to the analysis of sophisticated problem solvers because it supports a unified representation of both *meta-operators* [Davis, 1980, Genesereth, 1983, Genesereth and Smith, 1982, Hudlická and Lesser, 1984, Stefik, 1981, Wilensky, 1981] and domain processing. This unified representation views meta-operators

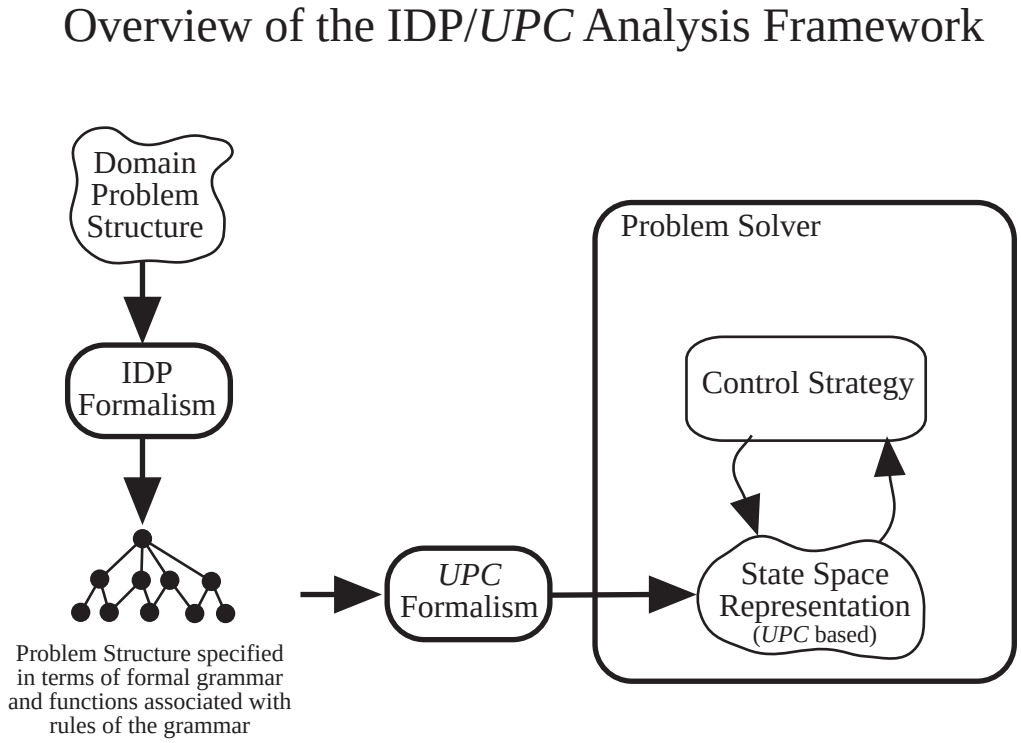


Figure 1.1. Overview of the IDP/*UPC* Framework for Analyzing Sophisticated Control

used in sophisticated control architectures as mechanisms that evaluate problem solving actions by selectively applying a process that examines a search path's relationship with other, possibly interacting, search paths [Lesser *et al.*, 1989b]. (Relationships are based on the distribution of domain events and are statistical in nature. For example, a relationship might indicate that two search paths lead to the same final result 50% of the time.) Furthermore, the process of examining a search path's relationships is viewed as a *distinct search operation* and not as part of the control architecture. Interrelationships between search paths are represented as abstract or approximate states that are explicitly created by search operators and not by a monolithic control process. Thus, the formalism can be used to explicitly analyze the tradeoffs between the effort a problem solver allocates to meta-level processing and the effort it allocates to base-level processing. In addition, this same analysis, or approximations based on it, can be incorporated in a problem solver's control mechanism to make dynamic decisions about when and how to use meta-level problem solving based on the emerging state of base-level problem solving. The success of the formalism in supporting this analysis is in large part due to the manner in which the representation of important relationships flows naturally from the formalism's representation of a domain.

The IDP/*UPC* framework is developed and demonstrated for a specific problem domain, interpretation, a specific class of problem instances from this domain, vehicle tracking, and a specific class of problem solving architecture, blackboard systems. As discussed by Carver [Carver and Lesser, 1991], the blackboard architecture was originally developed to deal with the difficult characteristics of a typical interpretation task: a very large search space; errorful or incomplete input data; and imprecise and/or incomplete problem-solving knowledge. These characteristics require a problem solving model that supports the incremental development of solutions, that can apply diverse types of knowledge, and that can adapt its strategies to the particular problem situation. The blackboard model has been popular for complex problems because it supports incremental problem solving and because it provides a great deal of flexibility in organizing the problem-solving process. For example, blackboards enable search-based problem solving that dynamically switches the abstraction level at which it works and reasoning techniques in which different kinds of search paths, including paths that are competing, cooperating, or independent, are pursued concurrently.

A blackboard system is composed of three main components, the blackboard, a set of *knowledge sources (KSs)*, and a *control mechanism*. The blackboard is a global database shared by all the KSs that contains the initial signal data and hypotheses, or partial solutions. A blackboard is typically composed of a hierarchy of levels and hypotheses are grouped into equivalence classes, each of which is associated with a particular blackboard level. The hypotheses are classified within the hierarchy based on their characteristic variables. The KSs embody the problem-solving knowledge of the system. KSs examine the state of the blackboard and create new hypotheses or modify existing ones when they are invoked. In the IDP/*UPC* framework, each production rule in IDP_i corresponds to a KS.

The control of blackboard problem solving is typically incremental, opportunistic, and sequential, or *agenda-based*. Incremental problem solvers construct interpretations on a piece by piece basis. Opportunistic problem solvers choose their next action by dynamically determining which potential action will allow the problem solver to make the most progress toward termination given the current situation. Sequential, agenda-based problem solvers must choose the next problem solving action to execute from a queue of potential actions. The typical control process is illustrated in Fig. 1.2. The execution of an operator generates hypotheses and new operator instantiations that can be applied to them. The new operators are placed on a queue and a scheduling or rating mechanism chooses the next operator to execute.

Control is one of the important issues that must be addressed in the successful formulation of Hearsay II type architectures and control continues to be an active area of research in the field of blackboard systems [Carver and Lesser, 1991]. The issues related to the design of a blackboard system's control component are often associated with the tradeoff between *domain processing* and *meta-level processing*. KSs that conduct domain processing extend search paths in an attempt to find a solution to a specific problem instance. KSs and control mechanisms that conduct meta-level processing attempt to somehow modify the problem solver's queue to improve the efficiency with which the domain processing is conducted. Meta-level control is required because a typical problem domain is too large to search exhaustively and still produce a timely result. Therefore, a portion of a problem solver's resources must be devoted to determining which KSs to invoke and which to eliminate from the agenda.

Since the introduction of the blackboard architecture, many research projects have focused on the development of control mechanisms that focus or limit search in such a way that the overall cost of problem solving is reduced and a consistent level of quality or correctness is either maintained or altered in a well-defined manner [Decker *et al.*, 1990, Garvey and Lesser, 1993].

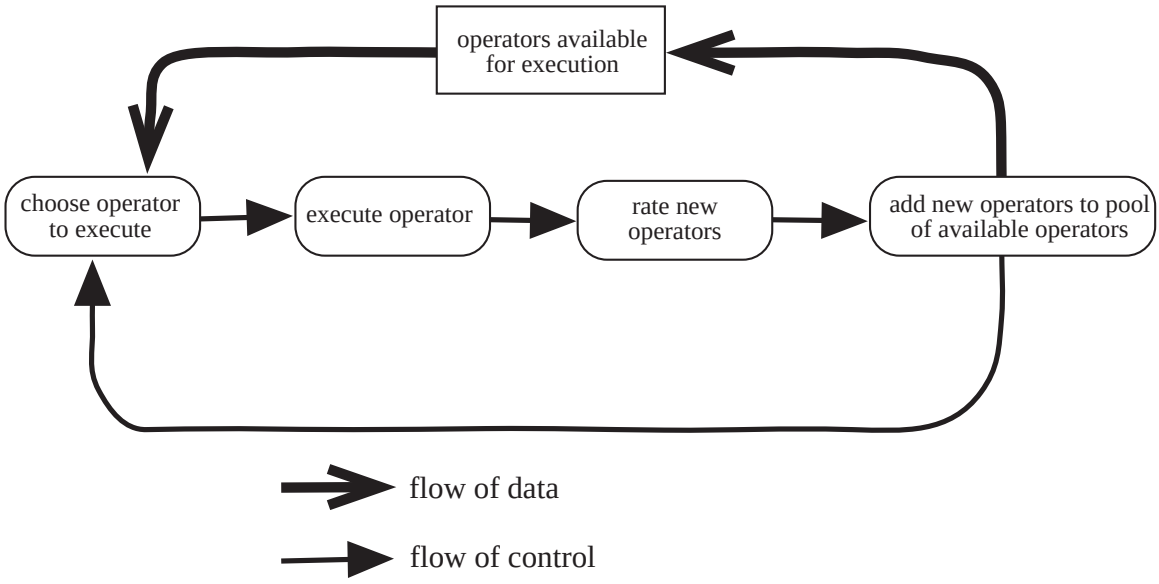


Figure 1.2. The Basic Control Cycle

Given the success of some "sophisticated" blackboard-based interpretation control mechanisms, there is an obvious desire to understand the underlying principles and domain properties in order to generalize the techniques to other domains. Such techniques include the focus of control mechanisms introduced in the Hearsay-II speech understanding system [Erman *et al.*, 1980, Hayes-Roth and Lesser, 1977] and the Distributed Vehicle Monitoring Testbed (DVMT) [Carver and Lesser, 1991, Corkill, 1983, Decker *et al.*, 1989], sophisticated control techniques such as goal processing [Corkill and Lesser, 1981, Corkill *et al.*, 1982, Lesser *et al.*, 1989a, Lesser *et al.*, 1989b], and abstracting and approximating computational domain theories [Decker *et al.*, 1990, Lesser and Pavlin, 1988].

Figure 1.3 represents a specific example of the class of blackboard problem solvers and the sophisticated control mechanisms studied in this thesis. The system shown in Fig. 1.3 is a high-level schematic for the integrated data-directed and goal-directed control architecture as implemented in the DVMT [Lesser and Corkill, 1983, Lesser *et al.*, 1987]. The basic blackboard architecture is modified to include a **goal blackboard** and a **goal processor**. The goal blackboard, which mirrors the data blackboard in dimensionality, contains goals representing **intentions** to create particular results on the data blackboard. Goals provide an abstraction over the potential actions for achieving a particular type of result and allow the system to reason about its intentions independently of the particular knowledge source (KS) actions at its disposal. The two general classes of goals are **data-directed** and **goal-directed**. The blackboard monitor uses domain knowledge to create data-directed goals in response to the addition or modification of hypotheses on the data blackboard. Each data-directed goal specifies the range of hypotheses that could result if the triggering hypotheses were extended at the same blackboard level or abstracted to the next higher level.

The creation of a goal does not guarantee sufficient information on the data blackboard to execute a KS to satisfy the goal, so the goal processor runs a **precondition procedure** for the applicable KSs to determine if there is sufficient data on the blackboard to successfully run the KS. When results indicate that a KS has sufficient information to satisfy the goal,

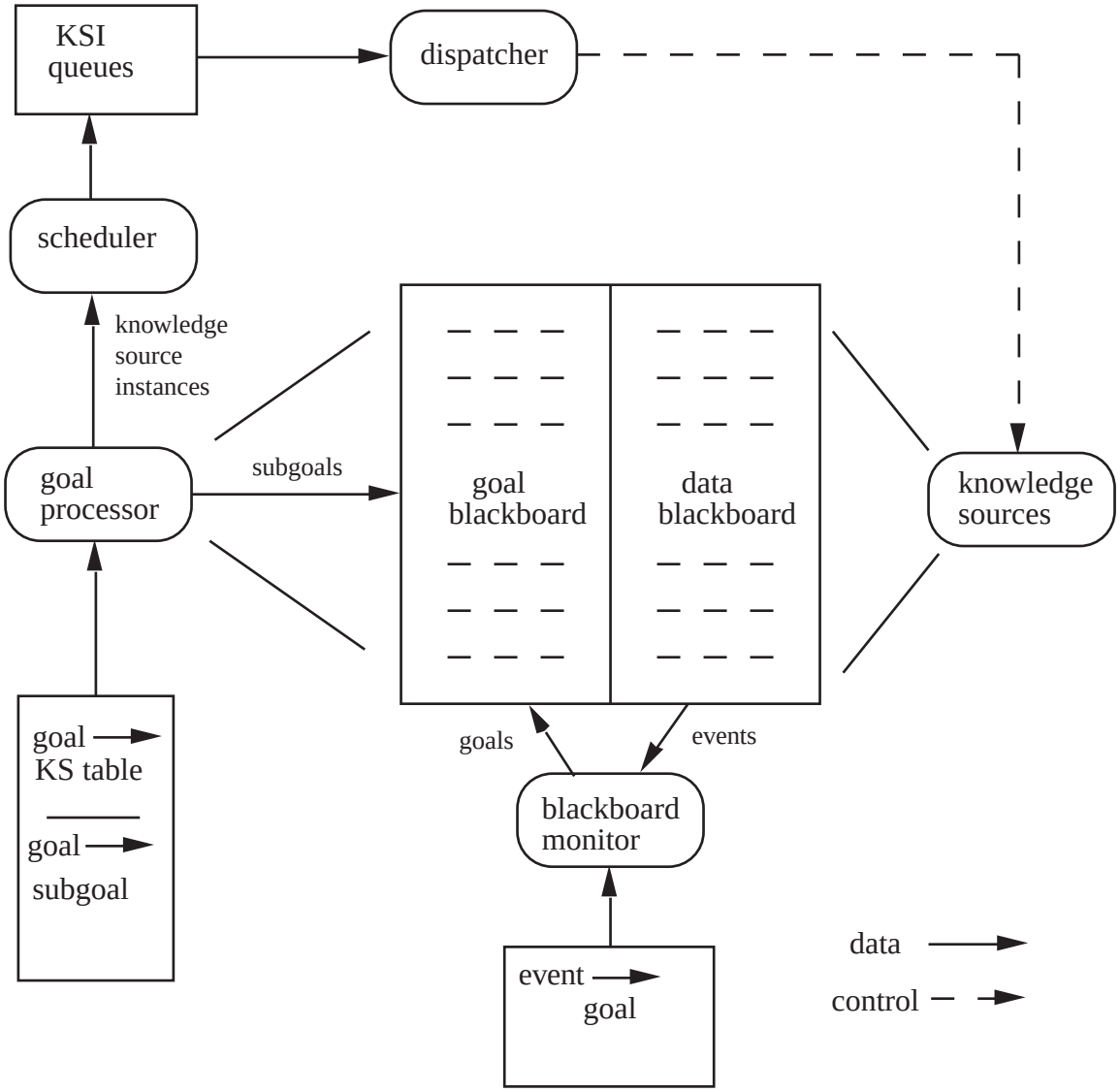


Figure 1.3. Integrated Data and Goal-Directed Control

the goal **triggers** a KS instantiation (KSI). The scheduler assigns the KSI a priority rating and places it on the **scheduling queue**. If sufficient information is not available to run a KSI, the goal processor creates **goal-directed goals** (subgoals) for driving up low-level data to be used to satisfy the original goal. Subgoals are rated the same as the original high-level, highly-rated goal. Thus, the ratings of low-level KSIs that could lead to satisfaction of the high-level goal will be increased.

This thesis studies data-directed and goal-directed control mechanisms, a related control mechanism referred to as approximate processing, precondition procedures, and bounding functions that prune search paths based on the characteristics of partial results. It is demonstrated that these mechanisms can be formally represented and that, when implemented as part of a problem solving system, their performance can be predicted and explained quantitatively. This demonstration is made in the context of a statistically optimal control strategy, used as an experimental control, and a heuristic control strategy. The following sections give more detailed introductions to each of the contributions made in this thesis and they provide a guide to the remainder of the document.

1.1 Defining Sophisticated Control, Interpretation Problems, and Complex Domains

Chapter 2 describes the theoretical foundations for this thesis and extends them by formally defining several important concepts. These extensions are a critical element of the IDP/*UPC* framework because they provide a perspective of problem solving where meta-level processing, such as meta-level abstractions and approximations, and domain processing can be viewed and analyzed from a unified perspective. The basis for this perspective involves the distinction made between *local, independent control architectures* (or simply *local control*) and *non-local, dependent control architectures* (or *sophisticated control*). In the IDP/*UPC* framework, control is defined as the "evaluate/expand" cycle shown in Fig. 1.2. At each step of problem solving, the problem solver's control component chooses the highest rated operator and executes it. The execution of the operator will usually cause other operators to be eligible for execution. For example, if a new state is created, the operators that can be applied to that state become eligible for execution. The control component's evaluation function rates the new operators and includes them in its deliberations in the next cycle.

As described in Chapter 2, when evaluating a search operator, local control architectures only consider the characteristics of the operator and the state(s) it modifies and not more global information such as the characteristics of other, possibly interacting, search operators. In contrast, sophisticated control architectures evaluate problem solving actions by selectively applying a process that examines a search path's relationship with other, possibly interacting, search paths [Lesser *et al.*, 1989b]. This process must be applied selectively in order to prevent a combinatorial increase in cost that would result from examining the relationships between every possible set of interacting search paths.

The analytical capabilities of the IDP/*UPC* framework are based on viewing the process of examining a search path's relationships as a *distinct search operation* and not as part of the control architecture. Interrelationships between search paths are represented as abstract or approximate states that are explicitly created by search operators and not by a monolithic control process. Furthermore, the process of examining the relationships between search paths is associated with an IDP domain theory representation consisting of an appropriate set of grammar rules and functions. Such problem solving actions have previously been referred to as

control problem solving, *meta-knowledge operations*, *meta-operations*, *approximate problem solving*, *abstract problem solving*, etc. In general, these problem solving actions will be referred to as *meta-operators*. They will also be referred to as *abstract* or *approximate operators* because they are primarily derived from abstractions of other operators. Thus, abstractions and approximations used in problem solving to explicitly examine relationships between search paths are represented as extensions of a basic IDP model and both primitive operators and meta-operators correspond to production rules of the associated IDP grammar representation. Chapter 3 defines some of the general IDP structures associated with the abstract problem solving actions that are studied.

Chapter 3 builds on the formal definitions of sophisticated control and complex domains by formally defining the class of problems we refer to as *interpretation decision problems (IDP)*. The IDP formalism models the structure of interpretation *domain theories* (a domain theory is the computational theory that is the basis for a problem solver's functionality) in terms of four feature structures: *component* (or *syntax*), *utility* (or *credibility*)[1], *probability* (or *distribution*), and *cost*.

The different feature structures that are defined by domain theories are combined into a unified representation by expressing them in terms of formal grammars and functions associated with production rules of the grammar. The formal grammar and functions associated with a domain theory will be referred to as the *domain grammar*. This unified approach allows search paths to be represented graphically as parse trees (or *interpretation trees*) of the grammar. By analyzing the statistical properties of the interpretation trees of a domain grammar, it will be possible to determine general characteristics of problem solving in the domain such as the expected cost for a random problem instance, the expected credibility of a solution, etc.

The statistical analysis is based on an approach where interpretation problems are viewed as discrete optimization problems, implying that the problem solver must consider, either implicitly or explicitly, every possible interpretation for a set of signal data and identify the best interpretation. This also implies that the problem solver must *connect* every path in the search space to either a dead end state or a final state representing a possible interpretation. (Connected paths are defined formally in Chapter 3.) It is important to note that the sophisticated control techniques that are the focus of this thesis can implicitly enumerate search spaces efficiently and that connecting a space does not necessarily require every potential final state to be generated. Viewing interpretation problems as discrete optimization problems sets the analysis framework apart from previous analysis techniques that are used to analyze problem solving in domains where the objective is to find the shortest, or lowest-cost search path, the highest-rated solution, or a solution path to a "winning position" [Pearl, 1984].

1.2 Formally Specifying Domain Structures and Problem Solving Architectures

Chapter 4 demonstrates how the IDP/*UPC* framework can formally represent problem domains and problem solving architectures. The IDP/*UPC* framework assumes that the phenomena, or individual problem instances, associated with a specific problem domain can be defined to occur in principled, or structured ways. Phenomena such as noise and missing data that make interpretation a difficult task do not "just occur." On the contrary, there are laws

[1] In Chapter 3, credibility structures are formally linked to the semantics associated with full and partial interpretations. Thus, a full or partial interpretation that has a high credibility can intuitively be thought of as having a highly consistent semantic interpretation and a full or partial interpretation that has a low credibility can intuitively be thought of as having an inconsistent or incomplete semantic interpretation.

Interpretation Grammar G'

grammar rule	distribution	credibility (utility)	cost
0.1 $S \rightarrow A$	$\psi(0.1) = 0.2$	$f_S(f_A)$	$g_S(g_A)$
0.2 $S \rightarrow B$	$\psi(0.2) = 0.2$	$f_S(f_B)$	$g_S(g_B)$
0.3 $S \rightarrow M$	$\psi(0.3) = 0.2$	$f_S(f_M)$	$g_S(g_M)$
0.4 $S \rightarrow N$	$\psi(0.4) = 0.2$	$f_S(f_N)$	$g_S(g_N)$
0.5 $S \rightarrow O$	$\psi(0.5) = 0.2$	$f_S(f_O)$	$g_S(g_O)$
1. $A \rightarrow CD$	$\psi(1) = 1$	$f_A(f_C,f_D, \Gamma_1(C,D))$	$g_A(g_C,g_D,C(\Gamma_1(C,D)))$
2. $B \rightarrow DEW$	$\psi(2) = 1$	$f_B(f_D,f_E,f_W, \Gamma_2(D,E,W))$	$g_B(g_D,g_E,g_W,C(\Gamma_2(D,E,W)))$
3.0 $C \rightarrow fg$	$\psi(3.0) = 0.5$	$f_C(f_f, f_g, \Gamma_{3.0}(f,g))$	$g_C(g_f, g_g,C(\Gamma_{3.0}(f,g)))$
3.1. $C \rightarrow fgq$	$\psi(3.1) = 0.5$	$f_C(f_f,f_g,f_q, \Gamma_{3.1}(f,g,q))$	$g_C(g_f,g_g,g_q,C(\Gamma_{3.1}(f,g,q)))$
4. $E \rightarrow jk$	$\psi(4) = 1$	$f_E(f_j,f_k, \Gamma_4(j,k))$	$g_E(g_j,g_k,C(\Gamma_4(j,k)))$
5.0 $D \rightarrow hi$	$\psi(5.0) = 0.5$	$f_D(f_h,f_i, \Gamma_{5.0}(h,i))$	$g_D(g_h,g_i,C(\Gamma_{5.0}(h,i)))$
5.1. $D \rightarrow rhi$	$\psi(5.1) = 0.5$	$f_D(f_r,f_h,f_i, \Gamma_{5.1}(r,h,i))$	$g_D(g_r,g_h,g_i,C(\Gamma_{5.1}(r,h,i)))$
6.0 $W \rightarrow xyz$	$\psi(6.0) = 0.5$	$f_W(f_x,f_y,f_z, \Gamma_{6.0}(x,y,z))$	$g_W(g_x,g_y,g_z,C(\Gamma_{6.0}(x,y,z)))$
6.1. $W \rightarrow xy$	$\psi(6.1) = 0.5$	$f_W(f_x,f_y, \Gamma_{6.1}(x,y))$	$g_W(g_x,g_y,C(\Gamma_{6.1}(x,y)))$
7. $f \rightarrow (s)$	$\psi(7) = 1$	$f_f(f_{(s)}, \Gamma_7(\ (s)\ ))$	$g_f(g_{(s)},C(\Gamma_7(\ (s)\ )))$
8. $j \rightarrow (s)$	$\psi(8) = 1$	$f_j(f_{(s)}, \Gamma_8(\ (s)\ ))$	$g_j(g_{(s)},C(\Gamma_8(\ (s)\ )))$
●	●	●	●
●	●	●	●
●	●	●	●

(s) = signal data $C(\Gamma_n(i,j,...))$ = cost of executing $\Gamma_n(i,j,...)$

Figure 1.4. Example of an IDP Grammar with Associated Functions

and principles that govern their occurrence and a problem solver can exploit these laws and principles in order to improve its performance. For example, a problem solver might be able to exploit its knowledge of certain recurring subproblems in a way that significantly reduces the overall cost of problem solving or that increases the quality of the solutions it generates. A domain's problem structure is a description of the causes of phenomena in the domain and knowing this structure is critical. For example, it is not enough to know that an interpretation domain experiences "noise" phenomena in order for a problem solver operating in that domain to successfully use a control architecture that was built to deal with noise in another, different domain. It is necessary to know if the characteristics of noise in the two domains are the same or similar enough that the control architecture can be extended to the new domain.

In IDP models, the different feature structures that are defined by domain theories are combined into a unified representation by expressing them in terms of formal grammars

and functions associated with production rules of the grammar. Nonterminals of the grammar represent intermediate problem solving states, terminal symbols represent raw sensor input, and the production rules of the grammar represent potential problem solving actions. The grammar rules of IDP models specify the component structure of a domain and each production, p, has associated cost and utility functions, g_p and f_p, that define the cost and utility structures. In addition, IDP models explicitly represent aspects of inherent uncertainty in a domain with the distribution function, ψ, that defines the probability structure of the domain. (i.e., ψ, along with other mechanisms, define inherent uncertainty in a domain.) For a given production, p, the frequency of the occurrence of p's right-hand-side (RHS) is specified by the distribution function $\psi(p)$. Thus, p can have multiple RHSs, RHS_1 through RHS_n, and the distribution of the RHSs is defined by $\psi(p)$. Finally, each production rule, p, is associated with a semantic function, Γ_p that is a function of the subtree components represented by the elements on the right-hand-side of p. Γ_p measures the "consistency" of the semantics of its input data and returns a value that is included in the credibility function. For example, in a speech understanding domain, Γ_p would rate the consistency of the meaning of a sentence and return a value indicating whether or not the sentence made any sense.

The framework involves the use of two distinct grammars that reflect the concept of a domain theory and an approximation of the domain theory used in problem solving. These grammars consist of a *generational grammar, IDP*$_G$, and an *interpretation grammar, IDP*$_I$. IDP$_G$ corresponds to the domain theory in the sense that it can be used to generate problem instances that correspond to the actual events that occur in a domain. IDP$_I$ is a representation of the problem solving actions available to a problem solver, including abstract and approximate operators used by the control mechanism. Thus, there is a clear connection between the structure of a problem domain and the associated problem solving architecture. Figure 1.4 is an example of an IDP grammar for a simple interpretation problem where IDP$_g$ and IDP$_i$ are equivalent. Included in the figure are the functions associated with the grammar rules that define the structure of the domain. The "interpretations" associated with an A, B, M, N, or O are considered final solutions in this example.

An important aspect of the IDP formalism is that it must be capable of modeling complex, real-world phenomena. Specifically, phenomena involving interactions over time and space. Intuitively, it may seem that the context-free IDP grammars might be inadequate representational tools. For example, consider a vehicle tracking domain in which a problem solver's task is to track multiple vehicles, some of which move in coordinated patterns. Such domains can be considered context-sensitive since the properties of some phenomena are dependent on those of another. As such, one would expect that the generative power of a context-free grammar would be inadequate to properly model the domain. This is a critical issue since the analysis tools in the IDP/*UPC* framework are dependent on the formal representation of the problem domain. If a domain cannot be accurately modeled, then it will not be possible to analyze it. In Chapter 4.7, we describe such a domain more fully and we discuss the techniques that can be used to construct IDP grammars capable of modeling context-sensitive elements of a problem domain. These techniques are primarily based on exploiting the *feature list* convention [Gazdar *et al.*, 1982, Knuth, 1968] presented in [Whitehair and Lesser, 1993].

Similarly, the analysis tools are also dependent on an accurate representation of the problem solving architecture and the relationships among potential problem solving operators. Chapters 4 and 10 present representations of problem solvers that use sophisticated mechanisms such as preconditions, pruning functions, a form of *goal processing* to focus problem solving activity more efficiently, and approximate processing. The control mechanisms modeled are based on

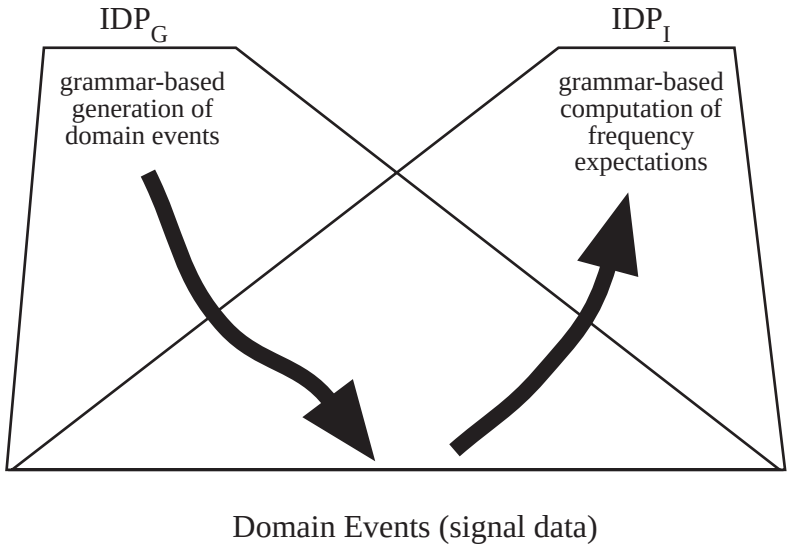


Figure 1.5. The Basic Approach to Calculating $E(C)$

techniques developed for the Hearsay II [Erman *et al.*, 1980], the DVMT [Corkill, 1983], and a real-time version of the DVMT [Decker *et al.*, 1990]. These chapters demonstrate how the IDP/*UPC* framework can represent and model sophisticated control mechanisms.

1.3 Defining Quantitative Analysis Tools and Methodologies

Chapter 5 defines a set of important quantitative analysis tools that can be used to predict and explain a problem solver's performance in a given domain. The most important of these is the complexity of a domain calculated in terms of the expected cost of problem solving for a specific problem instance, $E(C)$. $E(C)$ is measured in terms of computational cost and it represents the cumulative cost of applying all operators required to generate an interpretation. This is a general measure that has several advantages. It is intuitively easy to understand compared to other measures such as expected ambiguity, which is used in the calculation of $E(C)$. $E(C)$ can be used to compare both the performance of a problem solver across different domains or different problem solvers applied to the same domain with units of measure that are consistent. Most importantly, $E(C)$ represents what is probably the most significant aspect of a problem solver's performance. The definitions in Chapter 5 are verified in the experimental results presented in Chapters 7, 9 and 13.

The basic approach is a three step calculation. The first step calculates the expected frequency with which states are generated corresponding to each of the elements of the grammar. (Note that the set of all state frequencies is referred to as the frequency map of the domain.) This step relies primarily on the structure of the domain as specified in the grammar and the distributions associated with the rules of the grammar. The second stage calculates the expected probability with which paths from the states are pruned, which is called the pruning factor. This step relies both on the structure of the grammar and the domain's characteristics associated with the feature list. The final stage multiplies the expected frequency of path extensions (state frequency multiplied by pruning factor) by the expected cost of state expansion.

The general approach to calculating $E(C)$ is shown in Fig. 1.5. The generational grammar, IDP_g, is used to determine the statistical distributions for sets of signal data. These distributions, in turn, define the sample sets and the sample set weightings. The sample sets and weightings are used to calculate the statistical properties of groups of low-level domain events. These properties are then combined to determine the expected properties of higher-level results.

1.4 Formally Specifying the Structure of Search Spaces and Control Algorithms

Chapter 6 presents the basic *UPC* formalism for specifying the structure of a search space. The *UPC* formalism can be used as a basis for understanding the control decisions made by a problem solver and for explaining and predicting the effects of a control algorithm on a problem solver's performance. The *UPC* formalism maps IDP structures into a state space representation where, for each intermediate state in the search space, certain relationships between the state and the final states that can be reached from the state are represented explicitly. This is shown in Fig. 1.6 and in Fig. 1.7. For a given intermediate search state, s, that corresponds to a partial solution, and for each of the final states that can be reached along paths from s, the relationships that are represented include the expected cost of reaching each final state from s, the expected utility of each of the final states, and the expected probability of successfully reaching each of the final states. These expected values will be referred to as the *UPC values* for a state. *UPC* values are determined dynamically based on a perspective of problem solving that is local to a given state and on the statistical properties of a domain derived from a formal IDP specification. The calculation of *UPC* values for a specific state does not take into consideration the existence or absence of any other state. Both Fig. 1.6 and Fig. 1.7 illustrate how the representation of a state is extended by associating vectors of *UPC* values with the operators available to extend paths from the state.

The techniques that are used to analyze control architectures and, more generally, the performance of problem solving systems are based on formal IDP descriptions of the systems and on the resulting *UPC* values included in the extended representation of search states. As is discussed in Chapter 6, the utility, probability, and cost vectors associated with search operators can be derived from the corresponding IDP model. In essence, the IDP formalism explicitly represents the phenomena, such as functions defining distributions of domain events, sensor distortion, environmental noise, etc., that cause interpretation to be a complex task. These phenomena result in IDP representations that are structured. IDP structures are subsequently mapped to structures that appear in a search space and that can be exploited by the control component of a search-based problem solver. The *UPC* formalism explicitly represents these structures in a way that supports the analysis of a problem solving system's performance. (The use of the word "structure" will refer both to phenomena in IDP models and to corresponding phenomena in search spaces.)

The *UPC* formalism supports the analysis of problem domains and associated problem solvers by explicitly representing statistical properties of dynamic interactions among subproblems, by providing a framework for explicitly representing the cost and benefits of meta-level processing, and by modeling the dynamic structure of a specific problem instance based on the statistical characteristics of the problem domain. Together, these features enable the *UPC* formalism to explicitly model both the local and the more global aspects of a control decision in a manner that can be used as a basis for understanding the behavior of a problem solver and for explaining and predicting the effects of a control algorithm on a problem solver's performance. This is a necessary step in the development of general design theories for constructing

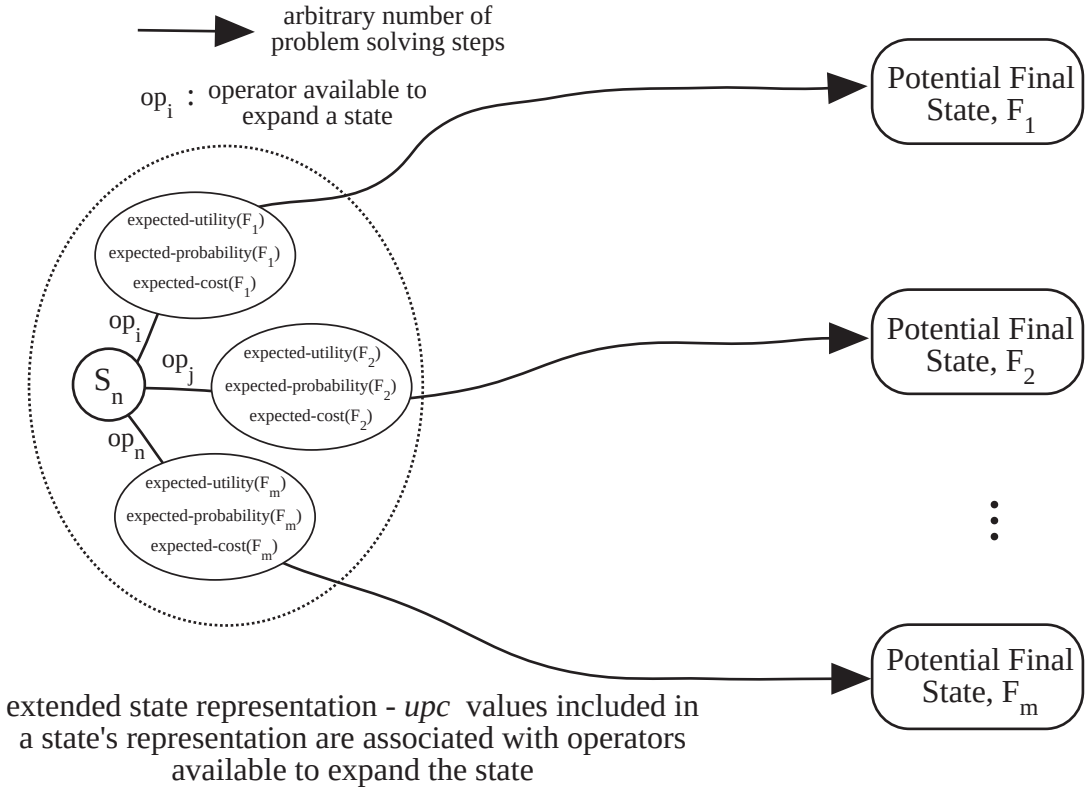
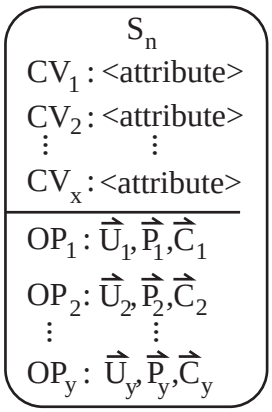


Figure 1.6. Derivation of *UPC* Values for a State



Extended State representation: operators applicable to each state are represented with the corresponding *Utility, Probability,* and *Cost* vectors.

Each vector entry consists of a measure of the *expected value* and a measure of the *variance.*

Expected values and variances are determined from IDP's cost and credibility functions. (In interpretation problems, credibility = utility.)

Figure 1.7. Representation of a Search State in the *UPC* Formalism

sophisticated problem solving systems such as blackboard systems [Carver and Lesser, 1991, Corkill, 1983, Erman *et al.*, 1980, Hayes-Roth and Lesser, 1977, Lesser *et al.*, 1989b].

1.5 Unifying the Representation of Meta-Level and Base-Level Processing

Chapter 8 defines a unifying representation of problem solving based on the *UPC* formalism that integrates meta-level and base-level processing in such a way that their costs and benefits can be directly compared. The *UPC* formalism provides a perspective of problem solving where the dynamic structure of a search space is explicitly represented and where meta-level processing and domain processing can be viewed and analyzed from a unified perspective. Interrelationships between search paths are represented as abstract or approximate states in *projection search spaces* or *projection spaces*. The abstract states are derived from a *base search space* defined by a formal model of a domain.

The base space contains no abstract or approximate states. In *UPC* models, meta-level processing and base-level processing are integrated into a unified representation where search paths connect states in the base search space with abstract states in projection spaces and the constraints associated with the abstract states are mapped back to the base space by explicitly altering the attributes of base-level search states to reflect the meta-level constraints. Consequently, sophisticated control mechanisms are incorporated in a state space representation where they can be viewed from the same perspective as traditional problem solving actions. This integrated perspective, shown in Fig. 1.8, suggests a new model of search where domain problem solving can be viewed as taking place incrementally and simultaneously and opportunistically in a continuum of abstraction spaces.

In the IDP/*UPC* analysis framework, projection spaces and their associated operators (including the projecting and refining operators) are viewed as *UPC* instantiations of the corresponding IDP domain theory model. The projecting, refining, and problem solving operators are represented as part of an IDP model of a domain theory. The component structure suggested by meta-operators is integrated into an IDP model as a set of grammar production rules and associated utility, cost, and distribution functions. Formulating problem solving in this way unifies two forms of problem solving, domain problem solving and meta-level control, that have sometimes been viewed as distinct classes. In addition, formulating meta-level control in this way allows a problem solver to determine abstraction levels dynamically or to alternate the level of abstraction at which problem solving occurs in order to opportunistically exploit the results of intermediate problem solving. The IDP/*UPC* framework is particularly effective for analyzing problem solving systems, such as the extended Hearsay-II [Erman *et al.*, 1980] blackboard model introduced by Lesser and Corkill, that integrate both *top-down* and *bottom-up* processing in a hierarchy of abstraction spaces [Corkill *et al.*, 1982, Corkill, 1983]. To our knowledge, no other analysis framework provides a perspective where different approaches to control can be analyzed as part of a unified domain theory.

In the IDP/*UPC* framework, by choosing the next problem solving action to perform, the control component determines the projection space in which problem solving will occur, and the operator which will carry out the action. For example, by choosing meta-operators, the control component projects one search space (possibly the base space) to another, more abstract space where certain subproblem interactions are explicitly represented. Alternatively, the control component can choose an operator that extends one or more partial solutions within a given projection space (i.e., carry out problem solving in a projection space), or it can "map back" (or *refine*) an abstract projection space to a less abstract space, possibly the

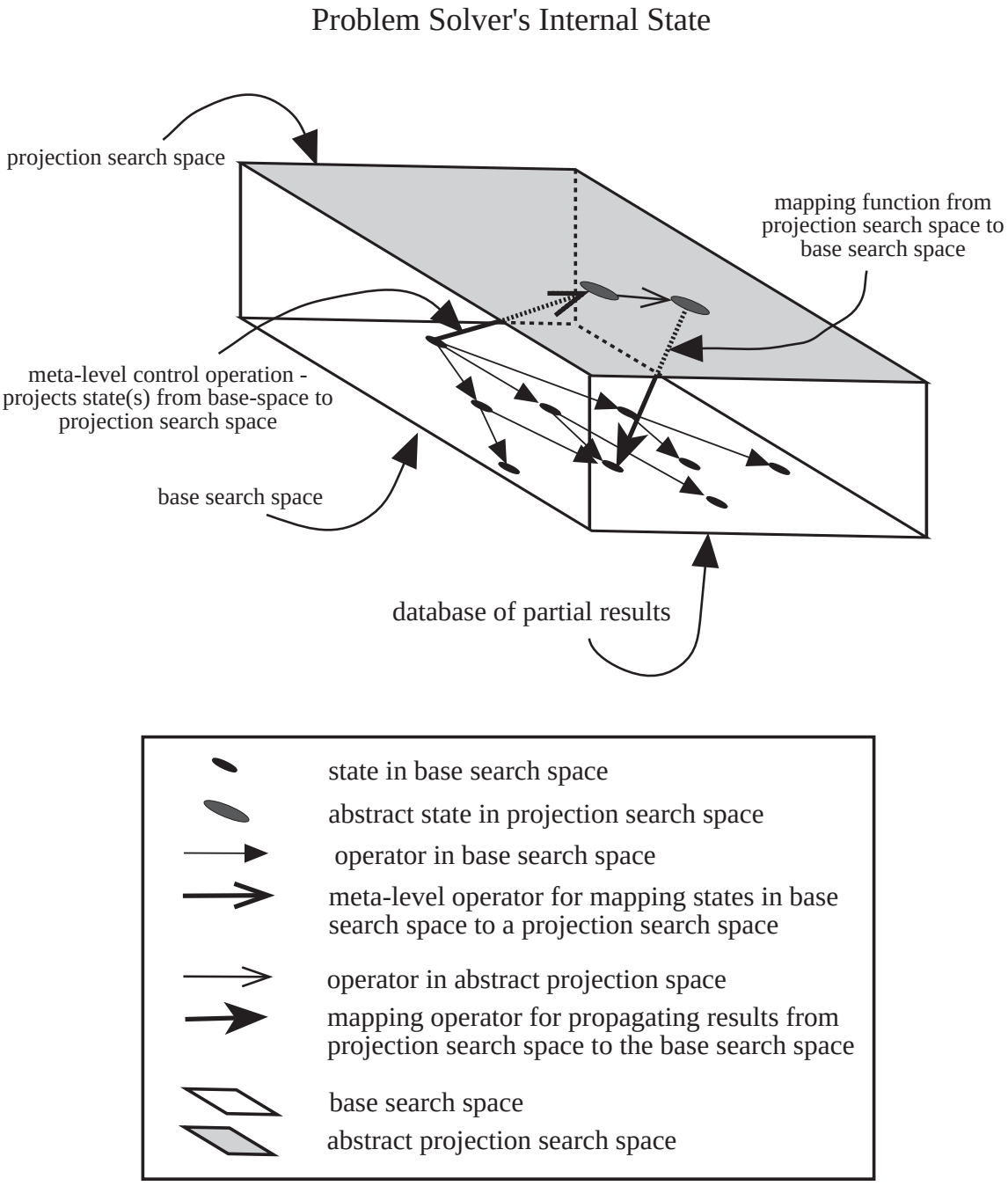


Figure 1.8. Explicitly Representing Meta-Level Control Actions as Components of a Problem Solver's Internal State

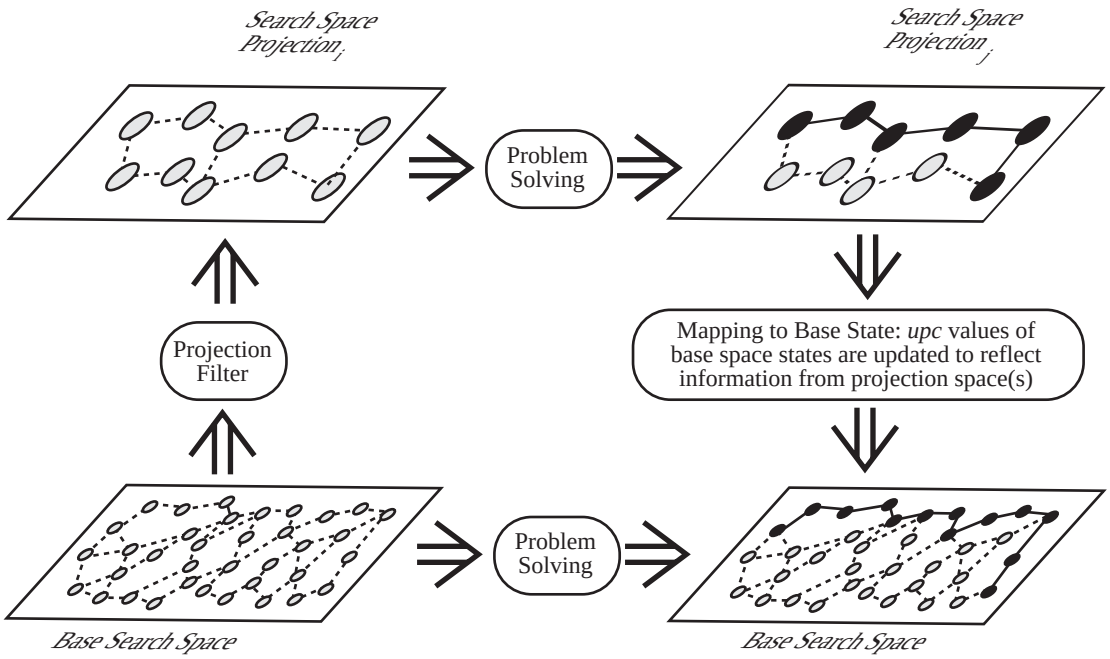


Figure 1.9. The Extended IDP/*UPC* Control Perspective

base space. Mapping an abstract state to the base space propagates the implications of any subproblem interactions back to the utility, probability, and cost values of the operators that can be applied to states in the base space. This perspective is illustrated in Fig. 1.9. To represent the relative worth of executing a meta-operator in a manner that can be used by a control component's evaluation function, a metric has been developed which is referred to as *potential*, for quantifying these relationships.

1.6 Introducing the Concept of Potential

Chapter 9 introduces and defines an important control concept, *potential* that is used in the analysis of the long-term costs and benefits of an action. The concept of potential is a critical element of the IDP/*UPC* analysis framework and it is used to address the question of how to evaluate the contribution made by meta-level control actions that use abstractions and approximations in terms that are consistent with the evaluation of problem solving actions that directly extend search paths in the base space. The concept of potential is applicable to all operators, but it is especially relevant to meta-operators. This is because, in general, meta-operators are not associated with effects that can be quantified in the same way as the effects of base space operators. The effects of meta-operators are related more to long-term reductions in problem solving cost or increases in solution quality. In contrast, the effects of base space operators are more closely associated with immediate effects resulting in the extension of base space search paths and they can be more easily quantified. Thus, potential will be used to develop mechanisms that support the understanding of the interrelationships that exist between the current set of states (i.e., the search paths that have been extended so far) and the states that can be derived from them. This includes using potential as a measure of the *expected long-term effects* an operator will have on problem solving in the base space. From

a statistical perspective, potential takes into account the changes in the *UPC* representation of base space states that occur as a result of the added information provided by a meta-operator. Thus, although a meta-operator may have no immediate effect on any base space search paths, which might appear to make it an undesirable choice of action, it may have a very significant long-term effect that reduces the expected cost of problem solving dramatically, making it a very good choice of action.

For example, there may be several operators available to extend paths from a state, s_n, to other states in the base space. In addition, there may be an operator available that will extend a path from s_n to an abstract state in a projection space. All of the base space operators may appear to be attractive in the sense that, from a local perspective, there appears to be a high-probability that the paths generated by the base space operators will eventually lead to final states with high credibilities. However, if the meta-operator is executed, it may generate an abstract state that indicates only one specific final state, F, is reachable from s_n. Thus, all of the operators that do not extend paths from s_n that might eventually reach F can be pruned. Another operator, which is called a mapping operator or mapping function, can then be executed to transfer this information back to the base space by modifying the *UPC* values of states in the base space to reflect that only the operators that can extend paths that might eventually reach F should be considered.

1.7 Establishing an Experimental Approach

Chapters 7 and 11 establishes a set of tools and a framework for experimental analysis and investigation of problem domains and problem solving architectures. One of the primary analytical uses of the IDP/*UPC* framework will be to compare and contrast the costs and benefits of alternative meta-operators in different domains. This will be achieved by defining domain independent *objective strategies* which are simple algorithmic statements such as "find the best solution," or "find the least cost solution," that will be used by a problem solver to make decisions regarding which action to take next. Inherent in the objective strategies used in the IDP/*UPC* framework is a perspective from which problem solving is viewed as an attempt by a problem solver to connect all the states in the base space by extending all potential search paths until they reach dead-ends or final states. (Objective strategies are formally defined in Chapter 6.4 and Appendix A.) In the IDP/*UPC* framework, knowledge-based actions that might otherwise be thought of as part of a control architecture are stripped out (leaving only the objective strategy) and represented in the same form as base-level problem solving actions. This enables direct comparisons to be made of the costs and benefits of alternative meta-operators in different domains. Thus, given a meta-operator, its expected performance characteristics can be identified for different domains, it can be compared and contrasted with other meta-operators, and classes of related meta-operators and domains can be identified. Example experiments are described in Chapters 7 and 9. A summary of these experiments is shown in Table 1.1. In the different experiments shown in the table, elements of the grammar representing meta-level control and characteristics of the domain are altered and the resulting problem solving performance is compared. In experiments 2 and 3, specific meta-control operators that are referred to as bounding functions are added to the grammar resulting in a decrease in the expected (and observed) cost of problem solving. In experiment 4, the mechanisms used to generate domain events were altered to generate more noise and missing data than originally expected. This resulted in an increase in the actual cost of problem solving. As shown in the table, using the IDP/*UPC* framework, certain performance characteristics,

Table 1.1. Example of IDP/*UPC* Experiments Comparing Alternative Meta-Level Control Architectures

Exp	Generation				Interpretation				Sig	% C
	G	Dist	U	$E(C)$	G	Dist	U	Avg. C		
1	1	even	0.5	201	1	even	0.5	203	N	100
2	2	even	0.5	189	2	even	0.5	187	N	80
3	3	even	0.5	180	3	even	0.5	181	N	80
4	1	skew	0.5	368	1	even	0.5	369	N	100
...	...	...	...	...	...	...	...	...	...	...

Abbreviations

Exp: Experiment
G: The problem solving grammar used;
1: G'
2: G' and bounding functions with cost 10,
3: G' and bounding functions with cost 1,
Dist: Distribution of Domain Events;
even: domain events evenly distributed
skew: distribution skewed to more credible events,
U: expected problem instance credibility;
0.5: problem instances have expected credibility 0.5
0.25: problem instances have expected credibility 0.25
0.75: problem instances have expected credibility 0.75
$E(C)$: Expected Cost of problem solving for given grammar
Avg. C: actual average cost for 1000 random problem instances
Sig: Whether or not the difference between expected cost and the actual average cost was statistically significant
Y: yes, there is a statistically significant difference
N: no, there is not a statistically significant difference
% Correct: percentage of correct answers found

such as the cost and quality of problem solving, can be predicted and and the predicted values and actual values can be compared. The basic control mechanism used to generate the experimental results is described in Chapter 6.4.

It is important to understand the philosophical underpinnings of the experiments that will be presented in this thesis. The IDP/*UPC* analysis framework is based on a philosophy emphasizing the importance of a problem domain's *structure*. At its simplest, this philosophy holds that the "AI universe" is structured and bound by laws and principles in much the same way that the "chemical universe" is structured and bound by laws expressed in the periodic table of elements or that the "physics universe" is structured and bound by the theory of relativity or by quantum theory.

As with the laws and principles that we associate with the physical sciences, e.g., the law of gravity, the theory of relativity, etc., the laws and principles discovered by AI scientists are meant to be applied to the models of reality that we construct to explain the natural world. In addition to forming the basis for continuing scientific research, these "models of reality" that scientists construct are subsequently used to develop design theories for building artifacts that will operate in the natural world, artifacts such as speech understanding systems, image understanding systems, etc. In both these endeavors, the key element is the model of reality that is constructed to explain and predict events in the natural world.

Thus, the IDP/*UPC* philosophy asserts that the occurrence of phenomena in any AI domain, not just the interpretation domains that are studied here, can be described formally and that this formal description is structured and constitutes the "causes" of the phenomena. Furthermore, the control architecture of a problem solver can exploit the structure of a domain's formal description in order to improve the performance of a problem solving system.

The primary contribution of the IDP/*UPC* analysis framework is that it provides a formalism for expressing the structure of the natural world in a way that can be used both for scientific analysis and for constructing design theories. In this thesis, both of these capabilities are demonstrated. In particular, it is shown that control and problem solving actions can be viewed from a unified perspective in terms of a problem domain's structure and it is shown that, for a given problem structure, theories can be constructed regulating the design of "meta" or "control" operators. Eventually, the IDP/*UPC* analysis framework could be used to develop a very broad perspective of problem solving that unifies diverse problem solving approaches including AI search techniques and techniques normally associated with operations research such as linear programming.

In the IDP/*UPC* framework, analysis of a problem solving system requires the explicit consideration of four elements: a problem's structure and a problem solver's *objective strategy*, *control architecture*, and *performance level (or behavior)*. In this thesis, a problem's structure is thought of as a patterned organization of the properties governing the creation of problem instances of a specific domain. The IDP formalism, introduced in Chapter 1, expresses structures in the form of phrase structured grammars with context-free production rules[2] and functions associated with rules of the grammar. Details of the IDP formalism are given in subsequent chapters.

Furthermore, in the IDP/*UPC* analysis framework, every problem solver will be associated with an objective strategy that defines the goals a problem solver is trying to achieve. For example, simple objective strategies include, "find the least cost solution," or "find a solution as quickly as possible," or "find the best solution." The objective strategy of a problem solver can be thought of as being analogous to the *objective function* of problem solving strategies such as the simplex algorithm [Papadimitriou and Steiglitz, 1982].

In a typical analysis situation, the object of consideration will be the control architecture – the algorithm(s) used by a problem solver to choose its next problem solving action. Control architectures will be expressed in terms of the problem structure. A given control architecture will be represented as rules (and associated functions) of the grammar used to define the problem domain.

In the IDP/*UPC* analysis framework, a problem solver's performance will primarily be measured in terms of the expected resources required to solve a problem and the expected

[2] The grammar representation used in the IDP/*UPC* framework is an extended form of the traditional context-free grammar representation. The extended form is used to explicitly represent information needed for analysis.

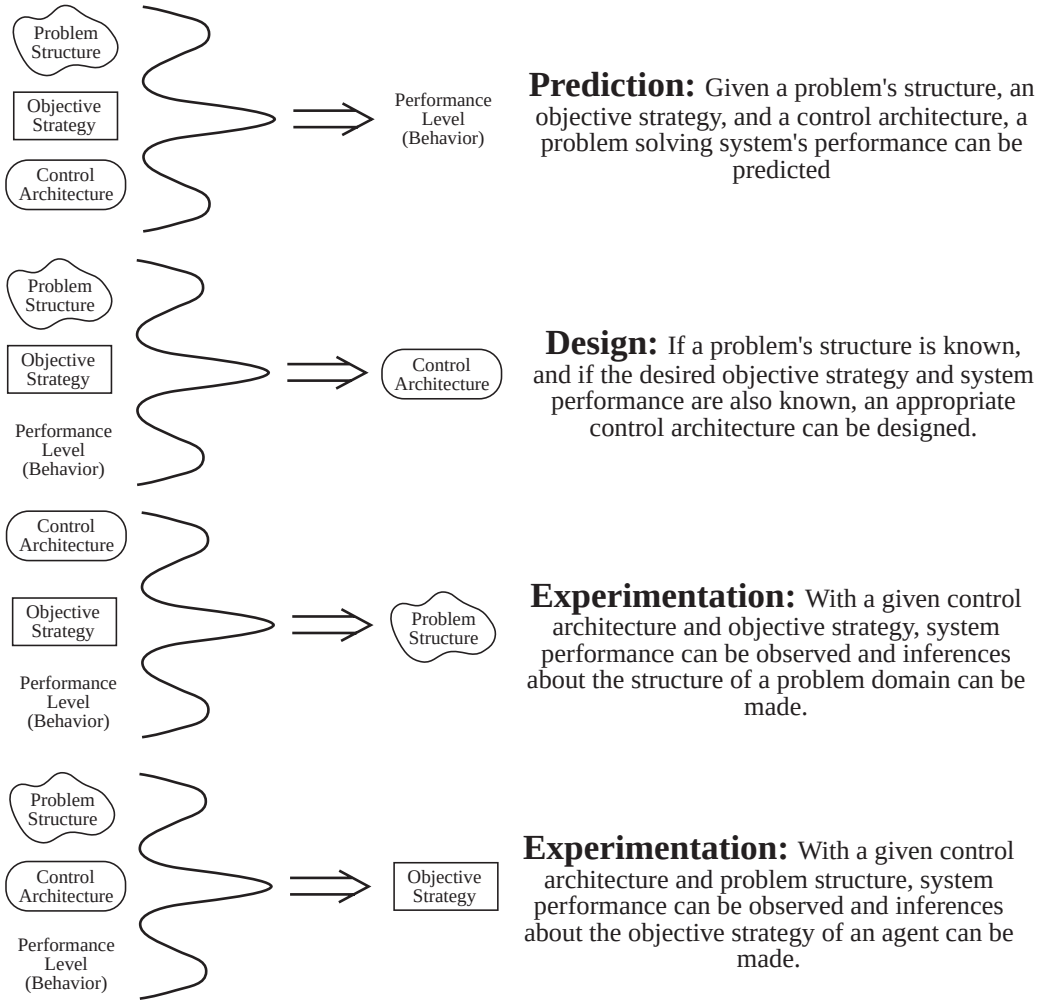


Figure 1.10. Summary of Analysis Perspectives Supported by the IDP*UPC* Framework.

"quality" of solution, where quality will normally be associated with a percentage of correct answers found in a series of test cases.

Though the primary emphasis of the analysis framework rests on the importance of a problem's structure, the significance of the other three elements should not be overlooked. In fact, the existence of these four elements implies a variety of analysis paradigms. These are summarized in Fig. 1.10, which illustrates the relationships between a problem's structure and a problem solver's objective strategy, control architecture, and performance level (or behavior). As shown in the figure, there are four basic analysis paradigms, *prediction*, *design*, *experimentation (problem structure)*, and *experimentation (agent objective strategy)*.

In the first paradigm, prediction, analysis focuses on predicting a problem solver's performance based on a given objective strategy, control architecture, and problem structure. In this thesis, we will demonstrate the validity of the IDP/*UPC* framework by showing that a problem solver's performance can be accurately predicted based on a formal analysis of the objective strategy, control architecture, and problem structure.

The ultimate objective of this work is to develop methodologies for constructing design theories. This objective is addressed by the second analysis paradigm, which will be referred to as the design paradigm. The design paradigm is not discussed in this thesis.

The two experimental analysis paradigms can be used to infer either a problem structure or an objective strategy given the other analysis elements. These forms of analysis will also be addressed in future work. For now, it is interesting to note that in the experimentation paradigms, the control architecture actually becomes an experimental tool rather than the object of analysis. In an experimental analysis paradigm, the control algorithm can be chosen or modified in order to determine the structure of a problem domain or to determine the objective strategy being used by a problem solving agent.

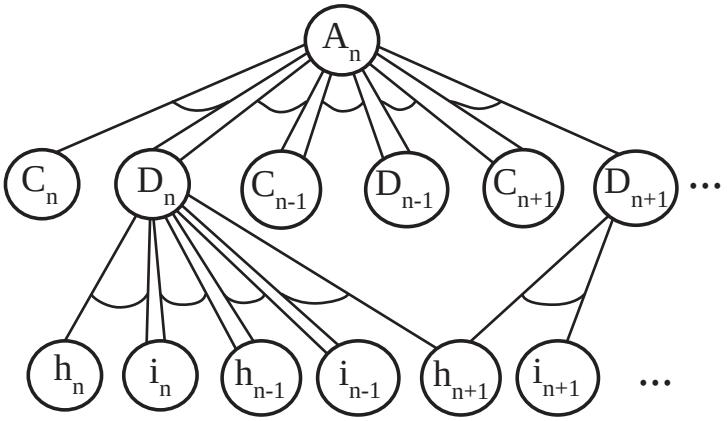
For example, a researcher may wish to investigate the structure of a particular image interpretation domain. The researcher may carry out a series of experiments in which she makes assumptions about the structure of the domain, embeds these structures into an IDP representation, and then designs a control architecture to exploit the assumed structure. By implementing both a simulation of a problem solver operating in the domain with the assumed structure as well as a real problem solver operating in the actual domain, the researcher will be able to conduct comparative studies to determine if the assumptions about the domain's structure of the image interpretation domain are reasonable.

1.8 Defining the Pruning, Preconditions, Goal Processing, and Approximate Processing Sophisticated Mechanisms

Chapter 10 formally defines a set of important sophisticated control mechanisms including preconditions, pruning operators, goal directed processing, and approximate processing. These mechanisms are all represented as extensions to a basic interpretation grammar. An example of the modifications that can be used to extend an interpretation grammar to represent meta-level problem solving actions is shown in figures 1.11 and 1.12. Figure 1.11 represents a typical interpretation grammar. The subscripts indicate that, for example, an A_n can be derived from a C_n and a D_n, a C_{n-1} and a D_n, a C_{n+1} and a D_n, a C_n and a D_{n-1}, etc. Given these rules, there is a great deal of ambiguity in this grammar. This grammar is based loosely on the vehicle tracking domain of the Distributed Vehicle Monitoring Testbed (DVMT) [Corkill, 1983]. Figure 1.12 shows the same grammar modified to include a class of meta-level operators referred to as *goal operators* presented in [Lesser *et al.*, 1989b]. This grammar is analyzed at length in Chapter 4.8.

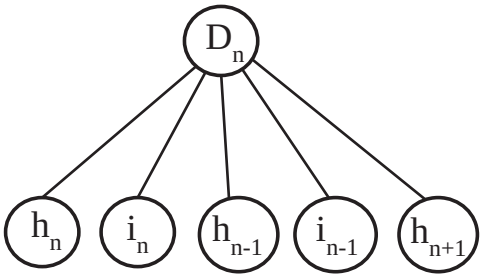
1. $A_n \rightarrow C_{n\pm1}D_{n\pm1}$
2. $B_n \rightarrow D_{n\pm1}E_{n\pm1}W_{n\pm1}$
3.a. $C_n \rightarrow f_{n\pm1}g_{n\pm1}$
3.b. $C_n \rightarrow f_{n\pm1}$
3.c. $C_n \rightarrow g_{n\pm1}$
4.a. $D_n \rightarrow h_{n\pm1}i_{n\pm1}$
4.b. $D_n \rightarrow h_{n\pm1}$
4.c. $D_n \rightarrow i_{n\pm1}$
5.a. $E_n \rightarrow k_{n\pm1}j_{n\pm1}$
5.b. $E_n \rightarrow k_{n\pm1}$
5.c. $E_n \rightarrow j_{n\pm1}$
6.a. $W_n \rightarrow x_{n\pm1}y_{n\pm1}z_{n\pm1}$
6.b. $W_n \rightarrow x_{n\pm1}y_{n\pm1}$
6.c. $W_n \rightarrow x_{n\pm1}z_{n\pm1}$
6.d. $W_n \rightarrow y_{n\pm1}z_{n\pm1}$
6.e. $W_n \rightarrow x_{n\pm1}$
6.f. $W_n \rightarrow y_{n\pm1}$
6.g. $W_n \rightarrow z_{n\pm1}$

a. extended interpretation grammar, G_n'



(signal data)

b. example structures from grammar



c. additional intermediate structures from grammar

Figure 1.11. Interpretation Grammar

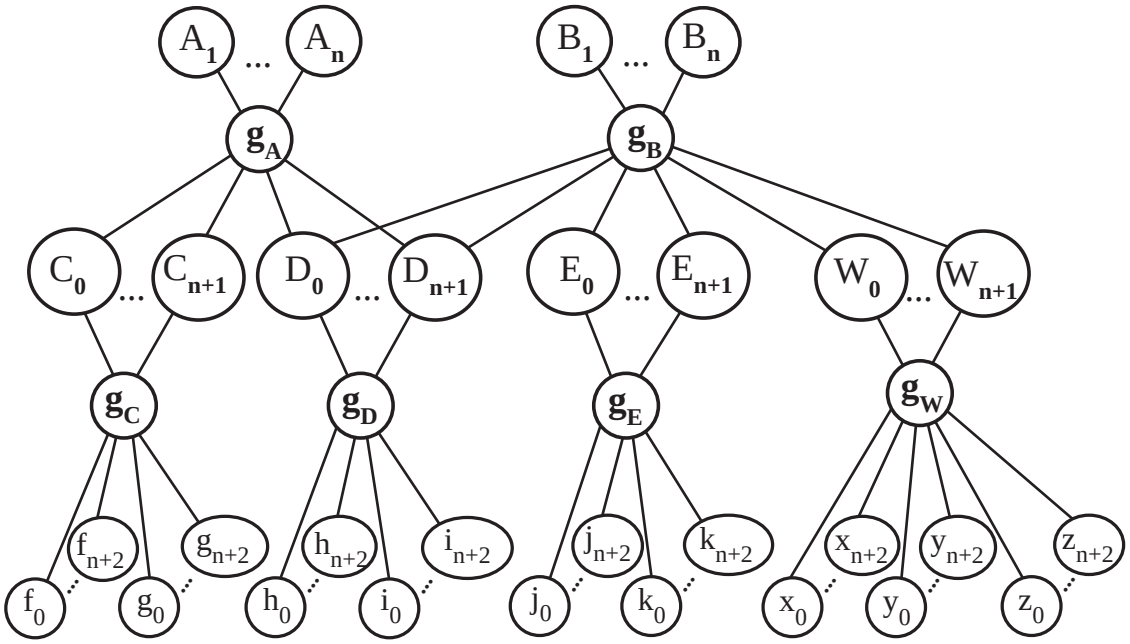


Figure 1.12. Representing Goal Processing in an Interpretation Grammar

1.9 Demonstrating the Analysis of Heuristic Control

Chapter 7 showed how the IDP and *UPC* formalisms could be used to correctly predict problem solver performance. The example problem solver architectures that were analyzed were based on control functions that used precise statistics about the likelihood of states reaching final states. We in part showed that this analysis could be extended to situations where precise statistics were not available. This was accomplished by using different grammars to represent the processes inherent in problem generation and the associated search processes used during problem solution. Specifically, the interpretation grammar did not have a statistically accurate view of the frequency of certain domain phenomena. In this thesis, it is shown that the analysis techniques are also appropriate for analyzing heuristic control techniques which are not based on using precise statistics. Specifically, Chapter 11 shows how these techniques can be applied to the analysis of two blackboard control architectures; one using a heuristic rating function based on level and credibility and heuristic termination criteria and another using heuristic rating based on level and credibility and information derived from a goal.

The heuristic rating function used is $LEVEL * RATING + POTENTIAL$, where $LEVEL$ is the blackboard level of the partial result an operator is attempting to extend, $RATING$ is the partial result's credibility, and $POTENTIAL$ is the operator's potential. Blackboard levels for the example domains are defined in Chapter 11.

The goal processing heuristic rating function is identical, except that, when a goal is created, all partial results, and the operators they triggered, subsumed by the goal have their ratings re-evaluated. The re-evaluation process compares the rating of the operator with a new rating determined by using the same heuristic rating function, but using information from the partial result that triggered the goal's creation, not the partial result that led to the original instantiation of the operator. If the new rating is higher, it replaces the old rating. If the new rating is lower than the old rating, no change is made.

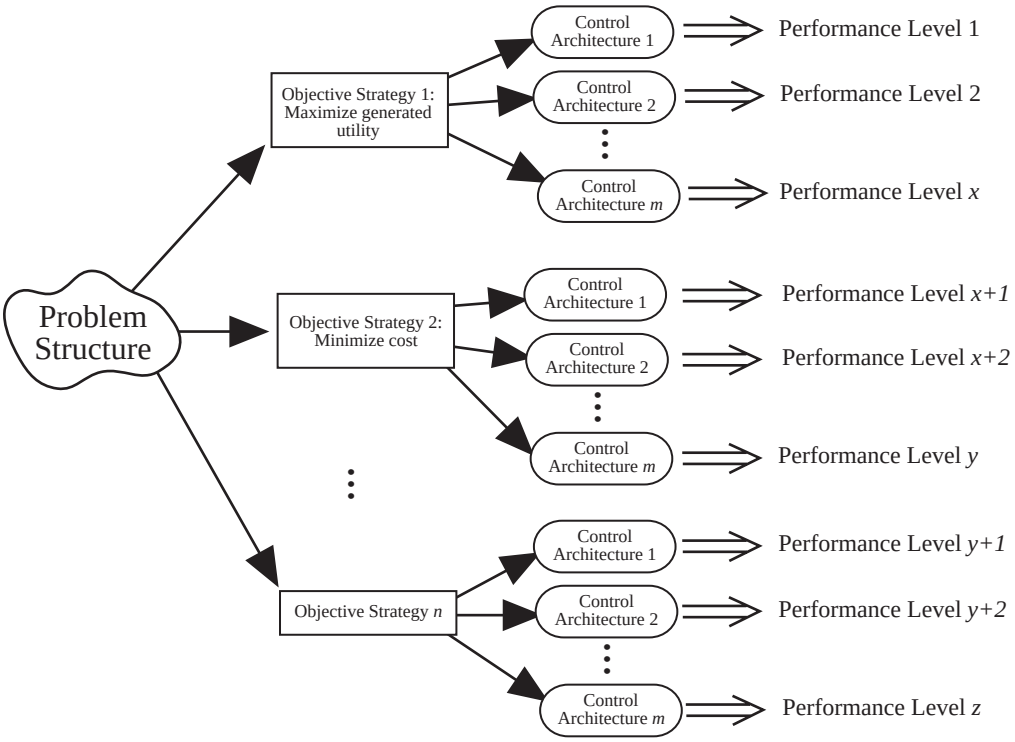


Figure 1.13. Illustration of the *Selection Problem* – A Given Problem Structure Implies Different Levels of Performance for Different Control Architectures

1.10 Defining Design Methodologies

Chapter 12.4 defines and demonstrates several design methodologies for constructing sophisticated problem solvers. The IDP/*UPC* framework provides a basis for establishing design theories for classes of AI search problems. The framework accomplishes this by clearly representing how assumptions about the characteristics of domain events and the structure of domain knowledge affect control architectures applied to the resulting search space. Furthermore, the framework analytically and empirically addresses two critical design issues, the *synthesis* problem and the *selection* problem. The synthesis problem, which involves the generation of approximations and abstractions for a given interpretation domain theory, corresponds to adding meta-operators (and associated functions) to the grammar component of a domain's IDP model. Chapter 4.8 presents examples of how the IDP formalism can be used to address the synthesis problem for a specific form of dynamic, hierarchical problem solving: goal processing.

The selection problem, which involves choosing approximations and abstractions for use in problem solving, is represented in Fig. 1.13. As shown in this figure, given a problem structure and an objective strategy such as "find the highest rated solution," or "find any solution as quickly as possible," the core thesis of this work is that different control architectures, when applied to the same problem structure, result in different levels of problem solving performance[3]. The IDP/*UPC* framework will address the selection problem by helping to identify the causal

[3] Problem solving performance can also be thought of *system behavior*.

relationships between problem structures, control architectures, and performance levels. This information can then be used to dynamically choose the appropriate control architecture based on the observed problem structure.

CHAPTER 2

RELATED RESEARCH

Heuristic search is an important Artificial Intelligence (AI) problem solving paradigm that, over the years, has been used as the foundation for a great variety of AI systems. The search paradigm is based on the concept of a space of *states* (or *nodes*) – data structures characterizing relevant features of the problem domain. (These data structures will be referred to as *characteristic variables* or *CVs* throughout this thesis.) Within this space, *search* is the process of "moving" through these states via domain specific *operators* that map states to states until a desired *final state* is found. Search invariably begins in a domain specific *start state* and as operators are applied "paths" are traversed between states. Search is often thought of as a process of *expanding* a state by applying one or more operators to it in order to generate new states. Each state, s_i, represents a *path* from the start state to s_i and expanding s_i is equivalent to extending the corresponding paths. A path that extends from the start state to a final state is referred to as a *solution* and paths that extend from the start state to intermediate states are sometimes referred to as *partial solution paths* or *partial solutions*. The general strategy for controlling the application of problem solving operators is based on the use of *evaluation functions*. As a state is expanded and new states created, an evaluation function gives each operator associated with the new states a rating that represents its relative worth or merit. The ratings are then used to determine an ordering for expanding paths.

Figure 2.1 is a representation of the internal state of a search-based, interpretation problem solver. As shown, the internal state of a problem solver is defined primarily in terms of the partial solution paths that it has expanded. The other components are the operators available for execution and the potential final states implied by the available operators. When a search path is created that represents a partial solution to a problem, at least one potential final solution is also created that, if generated, will include the partial solution. The importance of the implied potential final states is related to the analytical tools that will be defined and to the relationship between the analytical framework and previous work in search-based problem solving. Specifically, the concept of an implied final state is common to most search algorithms. For example, A* search is defined in terms of the final states implied in its characteristic equation $f^*(n) = g^*(n) + h^*(n)$. In this equation, $f^*(n)$ is an evaluation function for state n, $g^*(n)$ estimates the minimum cost path from the start state to n, and $h^*(n)$ represents an underestimate of the distance from n to a goal state (or final state).

If a partial solution is created and no corresponding potential final solution can be created, the path to the partial solution will be referred to as a "dead end" and all operators that can be applied to the partial solution can be eliminated from consideration. A "dead end" state is one for which one of the following conditions is true; if there are no operators that can be applied to extend paths from the state; or if the problem solver determines that there are no paths from the state that will reach a final state; or if the problem solver determines that there is no way that the final states that can be reached from the state will have the highest utility (credibility).

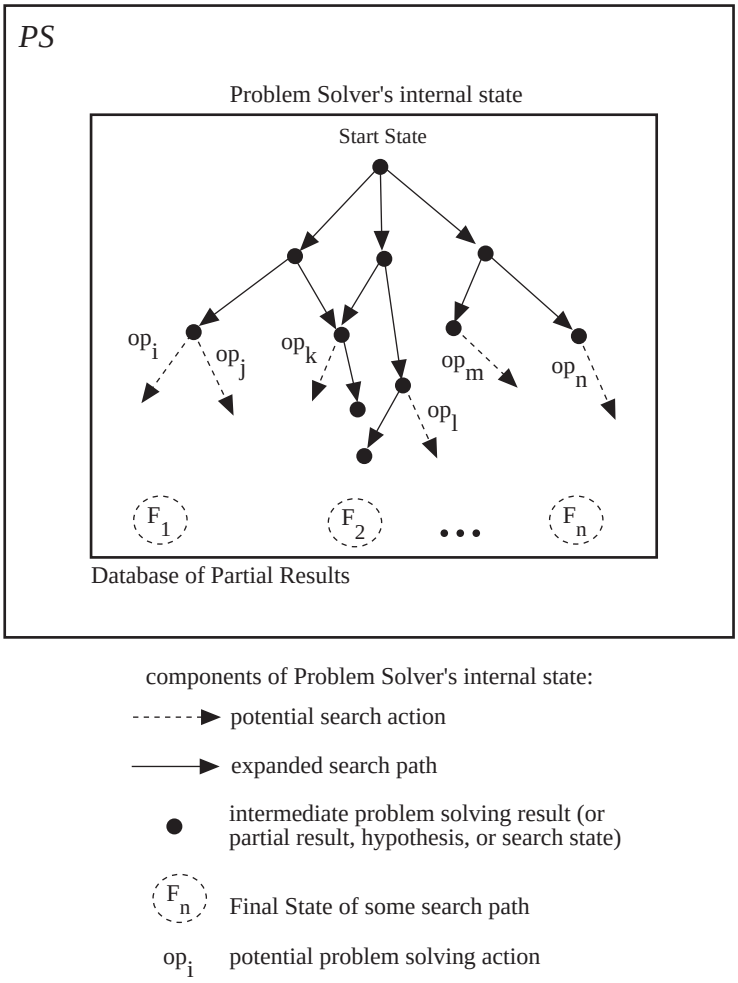


Figure 2.1. Representation of the State of the Problem Solver

When the internal state of a problem solver is such that all implied states have been generated and explicitly represented and there are no more operations that can be applied, the problem solver has reached *termination.* This is the same as saying that the base space has been connected – all paths from intermediate states have either been generated or terminated because the problem solver has determined that they cannot possible lead to the correct interpretation. It is also important to note that a problem solver, in trying to connect its internal state and reach termination, is also defining a search space. The analysis techniques will be associated with the search paths of a problem solver's internal state.

As a typical problem solver explores a search space by expanding states and extending partial solutions, the size of the search space, when measured in terms of the number of open solution paths plus the number of dead-end states plus the number of final states generated, grows at an exponential rate. Shown graphically, search spaces take on a tree-like structure where the start state is the root of the tree and the branches descending from each state represent the result of applying one or more operators to that state. In a typical problem, the branching factor ranges from only four or five to hundreds or thousands. Furthermore, a typical problem will require at least dozens of operator applications, resulting in dozens of levels to the "search tree." In general, the total size of a search space is calculated (approximately) by b^d where d is the depth of the tree and b is the branching factor of each state. (This calculation is approximate because the tree depth or branching factor could vary within a search tree.)

As a result of the enormous size of a typical AI search space, it is impractical or impossible to build problem solvers that are based on exhaustive search techniques. Instead, AI problem solvers employ strategies that selectively expand relatively small portions of the search space and then use these partial results to make inferences about the consequences of expanding much larger portions of the search space. These *implicit enumeration* strategies are often referred to as *control* or *meta-reasoning* strategies. (In this work, they will be referred to as meta-level, or abstract, operators.) The implementation of a control or meta-reasoning strategy will be referred to as a *control architecture* [Corkill and Lesser, 1981, Hayes-Roth and Hayes-Roth, 1979, Hayes-Roth, 1985].

The analysis framework is intended to be used to analyze *sophisticated control* architectures (sophisticated control will be fully defined in Chapter 2.2) in *complex domains* (complex domains are defined in the following section). Analysis will focus on architectures that use *abstract*, or *approximate*, reasoning mechanisms, which explicitly represent subproblem interactions, to implement efficient implicit enumeration strategies for interpretation problems.

The following sections will define and contrast complex domains and restricted domains. They will also define and contrast the associated sophisticated control architectures and local control architectures. Figure 2.2 summarizes these topics and illustrates their relationships to each other relative to *monotone* and *non-monotone* domains, which are also defined and discussed in the following sections.

2.1 Complex and Restricted Problem Domains

Early work on the search paradigm was restricted to constrained domains such as game playing [Berliner, 1979, Samuel, 1963], and theorem proving [Newell *et al.*, 1963]. The heuristic knowledge used in the search process was relatively limited. This can be expressed formally by saying that, for a restricted problem domain such as game playing or logic, $cost(operator_i) = O(1)$, i.e., the cost of applying an operator is a constant value. Thus, in a restricted domain, the cost of evaluating and expanding a single state is $O(1)$ and the cost

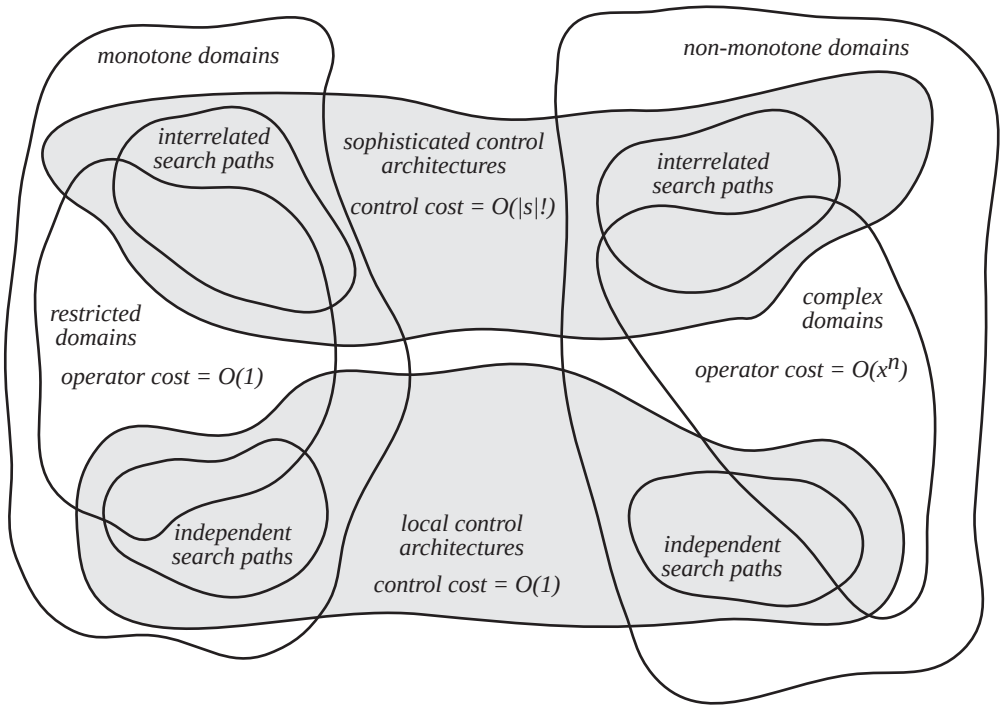


Figure 2.2. Classification of Problem Domains

associated with problem solving is a function of the number of times this constant cost must be incurred. Since there is little or nothing that can be done to reduce the incremental cost of expanding a single state, the most appropriate problem solving strategy is to reduce the number of states that are expanded by pruning paths wherever possible.

For the most part, the pruning strategy formed the basis for algorithms such as AO*, B*, SSS*, Alpha-Beta, and their generalizations [Berliner, 1979, Kumar and Kanal, 1988, Pearl, 1984, Stockman, 1979]. The strategy incorporated in these algorithms is based on the use of problem structures manifested in a problem solver's evaluation functions – ratings and knowledge of the search space's structure are used to determine which paths are most likely to lead to final states and which paths are dead ends. Paths that are more likely to lead to final states are expanded first, and the results are used to prune alternative paths and dead ends. In research projects involving restricted domains, the emphasis was on developing algorithms that enabled the problem solver to prune paths based on intermediate problem solving results [Berliner, 1979, Kumar and Kanal, 1988, Pearl, 1984, Stockman, 1979].

Previously, Pearl [Pearl, 1984] has shown that statistical properties of a problem solving technique, such as expected cost, can be determined from an analysis of the structure of a graphical representation of the search paths explored by the problem solver. In particular, Pearl examines the effects of pruning operators and heuristics for ordering the application of operators in various game playing domains and in other restricted domains. In contrast to the analysis techniques, the analysis techniques described by Pearl in [Pearl, 1984] do not take into consideration dynamic subproblem interactions or the long-term effects of an action.

Extended studies and analyses of the work on search techniques for restricted domains have resulted in the identification of a taxonomy of search procedures by Kanal and Kumar

[Kumar and Kanal, 1988]. Each of the procedural categories specified by Kanal and Kumar is defined in terms of the fundamental structure of the search spaces to which the procedures included in the category can be successfully applied. Kanal and Kumar's taxonomy links heuristic search techniques with problem solving techniques that are more closely associated with operations research (OR). This work has shown that basic AI search techniques can be classified as belonging to a subcategory of either *branch-and-bound* procedures or *dynamic programming* procedures [Kumar and Kanal, 1988].

More recently, AI problem solving has ventured into *complex domains* such as speech recognition, natural language processing, vision processing, pattern recognition, etc., that differ from the earlier, more restricted domains in two important ways. First, the search processes used in more complex domains do not exhibit the simplifying characteristics that would enable them to be categorized according to Kanal and Kumar's guidelines. This is in contrast to the characteristics of the restricted domains that imply the existence of certain types of search space structures that can be exploited during problem solving. These structures do not exist in the more complex domains and most of the search and pruning techniques that were developed in earlier work, such as AO*, B*, SSS*, Alpha-Beta, and their generalizations, are not applicable [Kumar and Kanal, 1988].

There are a variety of reasons for this. The most significant is that complex problem formulations do not exhibit *monotone* properties and structures. In a complex problem domain, certain characteristics of a search path are not guaranteed to increase (or decrease) monotonically, in relation to alternative paths, as the path is extended. For example, in a complex interpretation problem domain, as a path is extended, its credibility is not guaranteed to increase or decrease in a way that is monotone in relation to alternative paths. At any given point, the credibility of a path could go from "high-certainty" to "low-certainty," or vice–versa, while the credibility of alternative paths, which are extended with the same low-level data, could fluctuate in the opposite way. Kanal and Kumar's taxonomy only classifies problems that are, minimally, monotone. As discussed by Kanal and Kumar [Kumar and Kanal, 1988], in order for AO*, alpha-beta, B*, SSS* and their generalizations to be applicable to a given domain, the problem formulation for that domain must exhibit monotonic properties.

The second important distinction between restricted and complex domains is that the cost of applying an operator in a complex domain can be arbitrarily expensive, even exponential. This implies that the most efficient course of problem solving is not necessarily associated with path pruning. Instead, it might involve reducing the incremental cost of each search operator application, or replacing expensive operator applications with one or more inexpensive operators, etc. Consequently, the general strategies used to reduce problem solving costs in restricted domains must be modified or replaced when working with more complex domains.

2.2 Sophisticated and Local Control

Intuitively, local control architectures focus on ordering search activities and pruning paths based on the evaluation function ratings of intermediate problem solving results. Local control architectures are most relevant to problem formulations with well understood search space structures that limit the complexity of the knowledge and the amount of resources required to prune paths. Specifically, local control architectures are most useful in domains where search paths are independent of each other. In these domains, the merits of alternative paths can be computed by functions that are based solely on the local characteristics of the individual paths and the merits do not have to be recomputed when new states are added. (Path independence

is formally defined in Chapter 4.) The only interaction between paths occurs in the form of comparing their respective evaluation ratings. Because the evaluation functions are based solely on local information, their cost does not vary with the size of the search space. In fact, the evaluation functions can be thought of as constant cost functions. i.e., $\forall n, scost(f_{n,s}) = O(1)$, where $f_{n,s}$ is an evaluation function for the path represented by state n in search space s that defines the internal state of a problem solver at an intermediate state of problem solving. s corresponds to a problem solver's database of partial results shown in Fig. 2.1. As a consequence, scheduling and pruning algorithms based on these functions can be applied with little or no thought given to the costs involved. For example, given a search space with a very accurate and discriminating evaluation function, it may be possible to prune many search paths based on comparisons of evaluation ratings. In this domain, the control component of the search system could be a relatively simple mechanism that evaluates *every* state and expands those that are determined to be the "best" states and/or prunes states when it is determined that expanding them would be fruitless.

In contrast, sophisticated control architectures are most useful in situations where search paths are interrelated in such a way that extending one path somehow affects the results of extending other paths. The formal definition of a *relationship* between partial solution paths will be given in a later chapter, but for now it is sufficient to think of a relationship as a set of *constraints* that allow operators to function more efficiently. Constraints can also increase the likelihood that operators extend paths that will eventually lead to the correct solution and improve the ability of a problem solver to estimate the overall utility of a potential operator. For example, extending path A could reduce the cost of extending path B, or, perhaps, provide information that enables a better measure of the utility generated by extending path B.

Another way of thinking about this is to view the problem domain as consisting of a set of interdependent subproblems where solutions to subproblems are aggregated into an overall solution to the problem. From this perspective paths (or states) in the search space are considered to be *competing* when they lead to different solutions for the same subproblems, *independent* when they are solving subproblems that do not interact, and *cooperating* when they lead to solutions of subproblems where the solutions must be consistent with the solutions to a more comprehensive subproblem that includes the interacting subproblems as components. In the case of cooperating paths, potential solutions to a subproblem impose constraints on sibling subproblems. These ideas are very important in the IDP/*UPC* framework and a more extensive and formal description is discussed in Chapter 4 and in previous work of mine in [Lesser *et al.*, 1989b].

In choosing a path for extension in an interrelated search space, a sophisticated control algorithm seeks to optimize the amount of constraint that is generated by the extension both locally, for the path being extended, and more globally, for other paths related to the path being extended. The objective here is not necessarily to prune paths but to efficiently order problem solving activities so as to limit the overall cost of problem solving. Since paths are related to each other, the creation or extension of a specific path will (possibly) constrain future extensions of related paths. Thus, the goal of sophisticated control can be thought of as the optimization of constraint generation to minimize the cost of search by limiting the number of path extensions, as is done in conventional search, and by limiting the cost of individual path extensions and the cost of controlling the search process.

It is important to emphasize that constraining the expansion of a state is significantly different from pruning a path. For example, the constraint may be in the form of an ordering constraint, such as "do not expand state A until state B has been expanded." Such a constraint

may be beneficial in a situation where the results of expanding state B significantly reduce the cost of expanding state A. In this situation, it may be the case that no paths are pruned – the constraints generated by the expansion of state B simply reduce the resources needed by the operator that expands state A. Constraint can be manifested in the form of a delay. The problem solver may determine that it can delay certain problem solving activities in the hopes that subsequent actions will make the delayed activities meaningless and eligible for pruning. Though some of the search algorithms applied in restricted domains do delay certain problem solving actions, this delay is made with hope that other problem solving actions will allow the problem solver to prune the delayed action. The delay is not expected to reduce the cost of the action. In more complex domains, where an operator can have significant cost, reducing the cost of operator executions can be critical.

Optimizing control for constraint generation requires that analysis associated with the evaluation of a partially expanded path be based on existing alternative paths. This analysis will not be a constant cost function – the cost will vary based on the number and characteristics of existing paths. Furthermore, analyzing the relationships between paths could be a very costly computation and the incremental cost of analyzing additional path relationships as problem solving progresses could also be very high. Since the cost of evaluating a path is based partly on the number of other paths it might be related to, and since the total number of partially expanded paths grows exponentially, the cost of evaluating a single path in a complex domain could grow at a combinatorial rate. (i.e., $cost(f_{n,s}) = O(\mid s \mid !)$, where $f_{n,s}$ is an evaluation function for the path represented by state n in search space s, and $\mid s \mid$ represents the cardinality of s, i.e, the number of states expanded[1] so far and represented in s where s defines the internal state of a problem solver at an intermediate state of problem solving.)

[1] In evaluating a potential action, it may be necessary to understand its relationship to partial solution paths, and also to paths that have terminated in dead ends and paths that have been extended to final states. Paths leading to dead ends and final states may represent important relationships which indicate that the current path can be pruned, or that it is very important, etc.

Thus, the critical difference between sophisticated and local control is embodied in their respective costs. In a simple domain where solution paths are independent, the issue of control can be addressed by some control architecture that evaluates every potential path and uses the resulting information to efficiently expand and prune paths. It is usually assumed that the control component of such a problem solving system has cost = $O(1)$, the cost does not recur (i.e., paths are not re-evaluated in light of any new paths that are generated), and the problem solver need not be concerned with considerations associated with this cost. It is important to recognize that, even though evaluating a single path may have cost = $O(1)$, the number of paths that must be evaluated will grow exponentially. However, since the paths are independent, the addition of a new path does not affect the ratings of other, previously evaluated paths. Consequently, the incremental cost of evaluating the new paths that result from an expansion will still be $O(1)$. This form of search control, i.e., control architectures that are determined a priori, that are not modified based on intermediate problem solving results, that have cost $O(1)$, and that are not based on relationships between paths, will be referred to as *local, independent state evaluation control* or simply *local control.*

In more complex domains, the issue of control will require the use of a more costly evaluation mechanism and the problem solver must take this cost into consideration. This is a result of the fact that the cost of evaluating a single path is a function that grows at a rate that is potentially exponential in terms of the size of the search space. Given that paths in a complex domain are not independent, the expansion of a single path implies that *all* previously evaluated paths may need to be reevaluated, each at a potentially exponential cost. Because of the expense that may be incurred by these evaluations, it may not be practical to use a control mechanism that attempts to maximize the amount of constraint produced or seeks to find the "best" state for expansion. For example, if the cost of determining the "best" state for expansion has a cost that exceeds the cost of extending *all* the available paths, it is clearly more desirable to simply extend all the paths. Thus, it is no longer feasible to simply allow the control component to conduct exhaustive processing in order to determine which state to expand. Such a decision could be more costly than simply using exhaustive processing to solve the original search problem. This form of search control, i.e., control architectures that are determined dynamically based on intermediate problem solving results, that have a worst-case cost $O(|\ s\ |!)$, and that are based on relationships between partially expanded search paths, will be referred to as *non-local, dependent state evaluation control* or simply *non-local control.* Non-local control will also be referred to as *sophisticated control.*

As a consequence of sophisticated control, the problem solving process takes on a recursive quality where the control issue becomes a search problem in itself [Carver and Lesser, 1993]. The control component becomes a knowledge based mechanism that reasons and searches for the best operator. The control mechanism must reason about which states to evaluate, when to evaluate them, and which evaluation architectures to use. For this reason, control problem solving will be represented in the proposed formalism as an incremental search process that is based on the use of primitive operators and that is integrated with base-level processing in a way that allows the problem solver to reason about control versus domain processing. Previous research indicates that this is a reasonable perspective. For example, the BB1 system was implemented using a similar approach in which sophisticated control mechanisms were represented as problem solving operators that were explicitly considered by the scheduler [Johnson and Hayes-Roth, 1987].

2.3 Representing Complex Domains

The analysis framework depends on an explicit representation of the abstract and approximate search spaces that are used by a problem solvers meta-operators. Other researchers have reported analyses of the structures of abstract and approximate search spaces. For example, Knoblock discusses the maximum potential reduction in problem solving costs that can be achieved with the use of abstract and approximate search spaces [Knoblock, 1991b]. However, his analysis does not explain how these reductions might be achieved and his techniques do not address the issues presented in Chapter 1. Knoblock's analysis of the potential reduction of problem solving cost is related to the theoretical consideration of the effects of dividing a problem into a hierarchy of independent subproblems where determining solutions to all the subproblems results in a solution to the original problem. Knoblock then compares the cost of the solving the original problem, which is exponential in the size of the input, with the cost of solving the hierarchy of subproblems, each of which might also be exponential in the size of the input. Knoblock points out that, by subdividing a problem often enough, the actual cost of each subproblem is an exponential function of a very small number and, in fact, can be considered a linear or constant cost. Since the subproblems are all independent, the total cost of solving the hierarchy of subproblems is the sum of solving each of the individual components. Thus, assuming a reasonable hierarchical decomposition exists, the result is a cost that is linear in the size of the input. Knoblock does not, however, suggest a formalism capable of representing real-world problem domains or problem solving architectures in a manner that would support such a decomposition. Nor does it identify sophisticated control as a specific class of problem solving activity or relate the structure of a problem domain to the characteristics of a problem solver's performance. Consequently, though important for its theoretical developments and demonstrations, Knoblock's work does not create certain necessary elements that are required for constructing design theories.

In addition, the use of formal grammars and the associated graph structures as a basis for analyzing interpretation problems and for constructing problem solving systems is a common approach. One of the more significant uses of formal grammars is presented by Fu in [Fu, 1982]. Fu uses a representation he formalizes as a stochastic grammar that is superficially similar to the component structure of the IDP/*UPC* framework. However, Fu's work differs widely from this work in both representation and analysis. With respect to representation, the IDP/*UPC* framework includes extensions to the grammar not used by Fu. As defined in Chapter 3, an IDP grammar includes a set of functions associated with each rule of the grammar. One of these rule sets, the set of distribution functions, is consistent with Fu's distribution functions, but the others, the credibility and cost functions, are unique to the IDP formalism. In addition, the IDP formalism includes other extensions that differentiate it from Fu's work. An IDP includes the set of *solution non-terminals* (defined in Chapter 3) and the set of *singularities* (defined in Chapter 5.1.1) that are used to calculate statistical domain properties. The most significant distinction is that IDP grammars can represent both the characteristics of a problem domain and the structure of a problem solving architecture. In particular, an IDP grammar can be used to represent meta-level control actions available to a problem solver. In contrast, Fu's grammar are only used to represent the decomposition of design.

Furthermore, Fu's emphasis is on parsing, not on analysis or developing design theories. He does this by representing the semantics of a visual scene with a grammar. Thus, he encodes relationships between primitives and nonterminals with additions to the grammar. For example, he makes relationships such as "above," "next to," "inside of," etc., elements of the grammar.

Then he uses traditional parsing techniques to derive the semantics of the scene. He does not attempt to characterize the properties of a problem domain or a problem solving architecture as is done here.

In the analysis framework, the processes used to solve a problem are represented as a context free grammar. No claims are made about representing the semantics of a domain explicitly with the grammar. In fact, semantic properties are represented with function, Γ, that are arbitrarily complex functions. The approach used in the IDP/*UPC* framework is intended to formalize the subproblem relationships in such a way that statistical analysis can be done to determine general properties of the domain. These properties can be used be a problem solver's control component to effectively order activities.

Though the emphasis of Fu's work is quite different from the focus of this thesis, it is relevant to the future applications of this work. Fu's work demonstrates that real-world tasks can be accomplished using context-free grammars in form of parsing, or bottom-up processing. Other work indicates that context-free grammars can be exploited in a top-down manner as well. For example, impressive results have been achieved in automated design tasks by researchers such as Campbell [Campbell *et al.*, 1991] and Mullins and Rinderle [Mullins and Rinderle, 1991a, Mullins and Rinderle, 1991b] who use grammatical approaches to engineering design. In both cases, mechanical design tasks are solved by embedding knowledge about how to design certain artifacts in a grammar and then treating the design task as a form of natural language generation. The foundation for this approach was established by Knuth and the concept of an attribute grammar [Knuth, 1968]. These demonstrations of the power of context-free grammars are important to this work because they indicate that it is reasonable to expect that the IDP/*UPC* framework can be applied to real-world analysis in domains other than interpretation and that the framework's reliance on context-free grammars will not be overly restrictive in future work.

2.4 Related Research

Many previous research projects have sought to exploit a problem domain's structure in order to construct more efficient control architectures. These include projects such as ABSTRIPS, MOLGEN, and Hearsay-II. Although some of the work was not explicitly presented as being related to the structure of a problem domain, these previous efforts are relevant to the current project.

Work done by Fox, et. al [Fox, 1983], explicitly examines the structure of the problem domain. For problems defined as *constrained heuristic search* (CHS), i.e., problems that combine the process of constraint satisfaction with heuristic search, Fox identifies eight texture measures that define a problem's topology. These textures are used to dynamically characterize the search space in order to more efficiently focus problem solving. In two examples, experiments demonstrate a reduction of 25 – 80% in the number of search states generated during problem solving. Fox does not discuss the cost of dynamically computing texture measures nor does he present general algorithms for determining textures or their approximations. (Fox does present problem specific approximations for some of the texture measures, but there is no discussion of their generalization to other problem domains.)

In contrast to Fox's work on constraint satisfaction problems, the IDP/*UPC* framework is applicable to interpretation problems characterized by *constructive* search spaces. Constructive search spaces are identical to the *convergent* search spaces defined in Chapter 3.1 and similar to generate and test search spaces. Intuitively, constructive search spaces are characteristic of interpretation problems such as natural language understanding, image understanding,

sensor fusion, vehicle tracking, etc. Furthermore, Fox's work is based on problem domains with objective functions that assign final states equivalent utilities. Thus, all final states have identical worth and the problem solver's strategy is (usually) intended to find any final state as quickly as possible or with minimal cost.

The IDP/*UPC* framework is based on problem domains with objective functions that assign variable utility values to final states. The task in such domains can be stated in terms of finding the best, or most credible, answer, not just the first answer generated or the answer reached taking the shortest solution path. In other words, the first answer that a problem solver finds may not have the highest credibility of all the potential answers. Similarly, the least cost answer may not be associated with the answer that has the highest credibility. One class of such problems, *Interpretation Decision Problems*, is defined in Chapter 3.

The most important difference between Fox's work and the IDP/*UPC* framework is that the IDP/*UPC* framework attempts to model both short- and long-term structures. By short-term structures, we mean those structures that occur during the course of a single problem solving instance. Long-term structures are those that are characterized by statistical measures associated with *all* problem instances that can occur over time. The design theories derived from IDP/*UPC* representations will be applicable to the design of both general problem solving architectures and dynamic control architectures. Specifically, with respect to general architectures, the IDP/*UPC* framework can be used to address issues related to the *synthesis* and *selection* problems defined in Chapter 1. The synthesis problem, which involves the generation of approximations and abstractions for a given interpretation domain theory, corresponds to adding meta-operators (and associated functions) to the grammar component of a domain's IDP model. The selection problem involves choosing approximations and abstractions for use in problem solving. Thus, the framework can be used to design meta-level operators and the corresponding abstraction spaces and mapping operators that are to be included in a problem solving system. The framework can also be used to design the dynamic control algorithm (or evaluation function) that determines which operators to execute and which abstractions to use.

The IDP/*UPC* framework exploits a structural representation based on a statistical analysis of a problem domain. Fox's texture measures are related only to the dynamic characteristics of a specific problem solving instance. Though effective, it is not clear that they can be used to design problem solving architectures.

To summarize, the IDP/*UPC* research project is similar to Fox's work on CHS in that they both seek to formally define problem domain structures that can be used to more efficiently focus search-based problem solving. The works differ in that they are applicable to different types of search, CHS versus constructive search, and in the objective function used by the problem solver to rate final states. They also differ significantly in that Fox's texture measures are oriented more toward identifying and exploiting structures for increasing the effectiveness of dynamic control, and the IDP/*UPC* framework is oriented toward both dynamic control and the design of general problem solving architectures.

2.5 Generalizations

IDP/*UPC* based analysis tools are intended to be used to design, predict, and explain the performance characteristics of problem specific architectures and dynamic control strategies derived from the search-based control cycle shown in Fig. 2.3. The IDP/*UPC* framework is especially well-suited for the design and analysis of problem solvers that use abstractions and

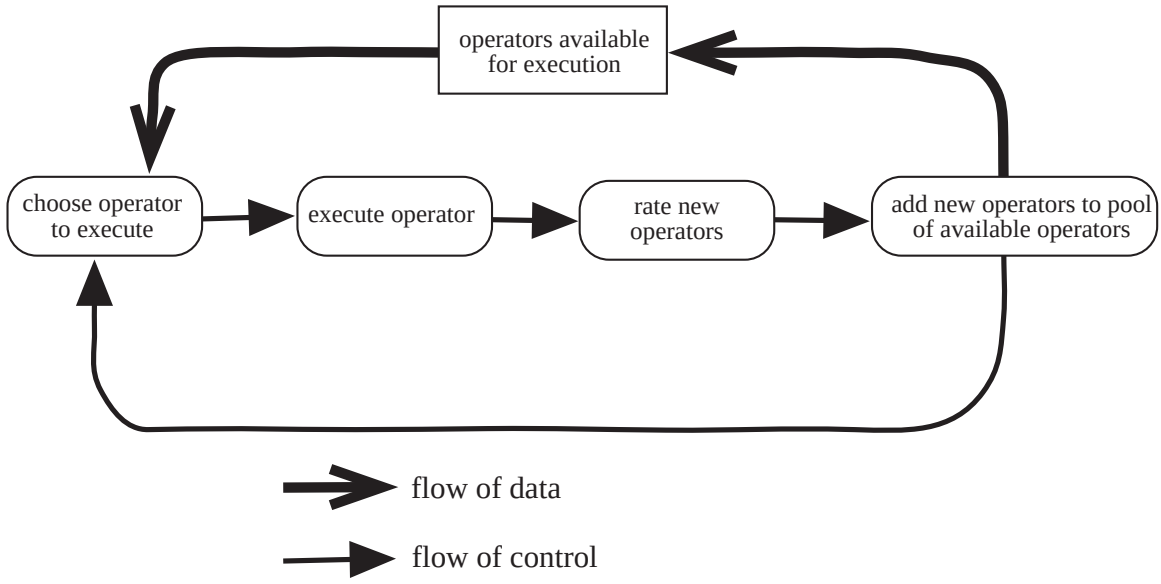


Figure 2.3. The Basic Control Cycle

approximations. Though the analysis framework is applied to the analysis of interpretation problems and constructive search spaces, its use is not restricted to these domains.

More generally, the framework can be applied to search-based problem solvers that use an evaluation function that rates prospective problem solving operations based on a decision theoretic computation. The computation must have input parameters that are functions of structures from the problem domain. For example, problem solvers with evaluation functions of the form shown in Equation 2.5.1.

Equation 2.5.1 $R(op_i(n_j)) = f_R(f_1(s^1_{op_i}) + c_1, f_2(s^2_{op_i}) + c_2, \ldots, f_m(s^o_{op_i}) + c_m),$

where $R(op_i)$ is the rating for the potential problem solving operator op_i applied to search state n_j, each f_k is a function of a domain structure represented by $s^k_{op_i}$, and c_i represents a constant. The representation $s^k_{op_i}$ should be interpreted, "the aspects of domain structure s^k relevant to op_i."

The evaluation function used in the experiments presented in Chapter 11 corresponds to this form. Specifically, for the evaluation function $LEVEL * RATING + POTENTIAL$, $f_1(s_n) = LEVEL(s_n) * RATING(s_n)$ and $f_2(s_n) = POTENTIAL(s_n)$. Then, $f_R(s_n) = f_1(s_n) + f_2(s_n)$, which is consistent with the required form.

It is important to note that Equation 2.5.1 is a representative example of the evaluation functions that are relevant to the analysis tools. The important aspect of this equation is the emphasis it places on the problem structures, $s^j_{n_i}$ and the absence of other factors. The specific form is not necessarily meant to specify restrictions for the set of problem solvers to which this work can be applied. For example, the analysis tools can be applied to evaluation functions where the f_j are not necessarily separable. Specifically, the analysis tools can also be applied if the computation of some f_j is dependent on the computation of another f_k, e.g., $R(op_i(n_j)) = f_R(f_1(s^1_{op_i}, f_2(s^2_{op_i})))$.

These restrictions are fairly limited and imply that the IDP/*UPC* framework can be applied to a broad range of search problems. The critical issue becomes the availability of definitions for the structures $s^j_{n_i}$. In Chapter 3, the structures that defined for interpretation problem domains include component (or syntax), utility, probability, and cost. These structures were chosen because they are common to the vast majority of interpretation problems and it is intuitively easy to understand their relevance. The structures used in an analysis are determined by the nature of the problem domain and the objectives of the problem solver and it would be incorrect to infer any restrictions from the specific characteristics or quantity of constraints used in the analysis work presented here.

The structures that are defined in Chapters 3, 4 and elsewhere in this thesis are based on the use of grammars and are applied to interpretation decision problems, or IDPs, as defined in Chapter 3. IDPs will be characterized as discrete optimization problems where the set of potential solutions could be generated with grammars. Thus, the structures that are defined are naturally derived from the specification of the solution space. The set of discrete optimization problems contains an extremely broad class of problems including many of the most important classes of problems studied in other disciplines such as operations research. This suggests that the applicability of this work is limited by the extent to which a problem's solution space can be represented as the language generated by a grammar. Also, the IDP/*UPC* framework extends the formalization work of others, most notably Kanal and Kumar [Kumar and Kanal, 1988] and, in so doing, helps integrate problems from artificial intelligence with operations research problems in a unified framework.

As will be discussed in Chapter 6.4, given appropriate definitions of problem structures such as utility, probability, and cost, it is possible to define simple control mechanisms that seek to optimize ratios such as (generated utility)/cost. With the necessary structure definitions, it should be possible to define many different decision theoretic control strategies.

The problem structures introduced in Chapter 4 are defined in terms of context-free, phrase structured grammars and functions associated with production rules of the grammar. As Chapter 4.7 discusses, this technique is well-suited for the study of interpretation problems. Though the use of grammars is a key element of the IDP formalism and a generally applicable methodology for specifying problem structures, is not necessarily required for extensions to the analysis framework. At present, experiments with other methodologies for specifying problem structures have not been conducted. However, it is certain that such methodologies exist and can be very effective in interpretation and other domains.

2.6 Chapter Summary

This chapter defines the class of problems referred to in this thesis as *sophisticated control problems* and the class of *complex problem domains* and relates both to previous problem and domain definitions. The resulting taxonomy is shown in Fig. 2.2. Sophisticated control and complex domains are formally defined in the context of search problems. Specifically, sophisticated control problems are those where, in order to find the statistically optimal operator to execute, the control mechanism must examine the relationships between all possible sets of partial search paths. Complex domains are search problems in which the cost of applying a single operator is potentially exponential with respect to the input to the operator and where the functions for extending search paths are non-monotonic. These definitions establish a formal specification for an important set of AI research problems, they allow these problems to be characterized more precisely, and they enable the results from research on these AI problems

to be compared and contrasted to the results of other research efforts in AI and to work in other fields, such as operations research, that have developed search-based formalisms. This chapter also defines the related concepts of *restricted problem domains*, in which search operators have a constant cost, and *local control*, which does not examine any interrelationships between potential problem solving actions.

CHAPTER 3

INTERPRETATION PROBLEMS AND THE IDP FORMALISM

The IDP formalism describes a class of problems, *Interpretation Problems*, in terms of a search process where, given an input string X, a problem solver attempts to find the most *credible* (or "best") explanation for X. Thus, tasks where a stream of input data is analyzed and an explanation is postulated as to what domain events occurred to generate the signal data are thought of as interpretation problems – the problem solver is attempting to interpret the signal data and determine what caused it. Interpretation is a form of *constructive problem solving* based on *abductive inferencing* [III, 1990]. Interpretation is similar to a closely related form of problem solving, *classification*, and to a more distant form of problem solving, *parsing*.

More formally, let an IDP be defined as follows:

Definition 3.1 Interpretation Decision Problem (IDP) - Given X, an arbitrarily complex input signal, determine the *best* element, e, of the discrete set of all valid interpretations, I, such that $\forall i \in I, f(e) \geq f(i)$, for evaluation function f, where a valid interpretation is one that explains all the elements, $x_j \in X$.

In Chapter 6.4 a variant of the Interpretation Decision Problem is introduced that can result in forms of problem solving that generate solutions that do not necessarily have the highest credibility rating. In this variant, the cost of problem solving can be significantly lower than the corresponding cost of problem solving in an identical IDP problem instance.

It is important to emphasize that, in interpretation problems, the set I is constructed dynamically. This is in contrast to classification problems where I is preenumerated and the problem solving task involves only the identification of the best element of I. Figure 3.1 is a representation of an Interpretation Decision Problem.

In most interpretation problems of interest, I is a potentially enormous, even infinite, set. In many of these domains, I can be specified in a naturally structured way and it is this structure that is exploited by control architectures to reduce the number of elements of I that must be generated or to otherwise increase the efficiency with which I is specified. Of particular interest are situations where I is finite and can be defined as the language generated by a grammar, G, and where the evaluation function, f, used during problem solving is recursively defined for strings i in I. This approach is similar to that used in the *Composite Decision Process (CDP)* model of Kanal and Kumar [Kumar and Kanal, 1988].

In these situations, interpretations take the form of derivation trees of X and the constructive search operators used in interpretation problems are viewed as production rules of G. G, therefore, defines how interpretations are decomposed. For example, the production rule $p \rightarrow n_1 n_2 n_3$ might correspond to the interpretation "p is composed of an n_1, an n_2, and an n_3." Given an n_1, an n_2, or an n_3, a problem solver may invoke the search operator o_p to try and generate a p.

Semantics associated with each production rule determine the actual domain interpretation. The semantic functions will typically make use of grammar element attributes that are not used

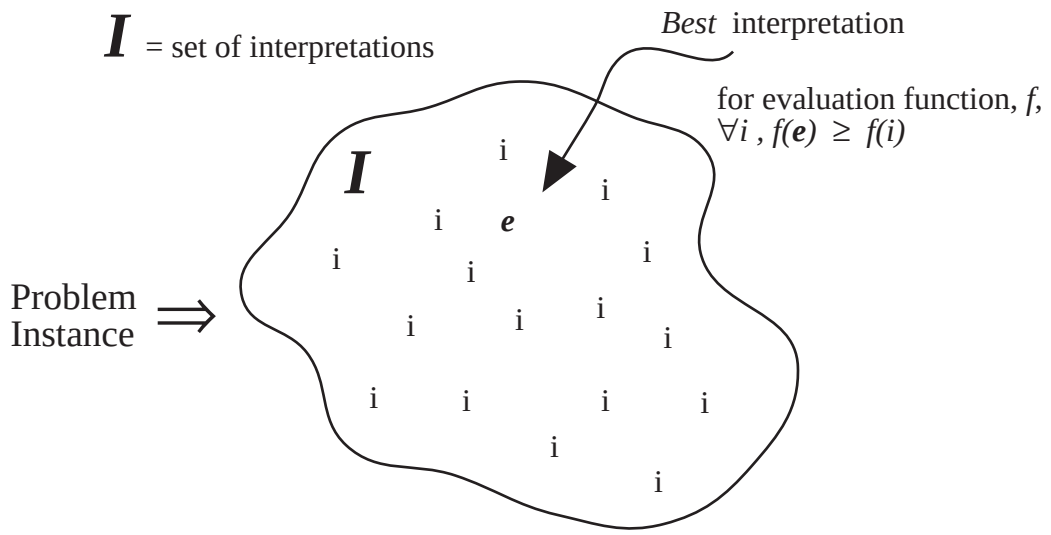


Figure 3.1. Representation of an Interpretation Decision Problem

by syntactic functions. These attributes will be represented using the *feature list convention* described by Gazdar, et al.[Gazdar *et al.*, 1982] and similar to the attribute grammars of Knuth [Knuth, 1968]. This is an important point. In interpretation problems, each syntactic rule of the grammar is associated with a corresponding semantic process. (These semantic processes are discussed further in Chapter 3.2.) Thus, during interpretation, each application of a search operator consists of a semantic as well as a syntactic operation. Furthermore, the functionality of the semantic processes is unrestricted and could include virtually any form of processing including production rules, neural networks, etc. For example, a semantic process in a natural language interpretation task might combine a noun phrase and a verb phrase into a sentence using a very complex process that simultaneously verifies consistency and combines the meanings of the two component phrases. Alternatively, phonemes might be combined in the same domain into words based on neural network algorithms.

3.1 Convergent Search Spaces

In the IDP formalism, the *primitive operators* available to solve a specific domain problem are represented as production rules of the domain's characteristic grammar. The mapping of interpretation decomposition to a formal grammar results in a type of search space that will be referred to as a *convergent search space*. The significance of convergent search spaces is that the relationships between search paths can be determined and analyzed based on a formal representation of a given domain. This analysis can be used to determine closed form expressions for the expected costs of problem solving. This is in contrast to domains where the relationships between search paths cannot be determined or formally analyzed.

Common to interpretation problems, convergent search spaces consist of search states that correspond to the symbols of the formal grammar's terminal and nonterminal alphabets. This is an important distinction because it limits the scope of each search state in a convergent space to an incomplete view of the search state relative to a more comprehensive, or global, view of

problem solving in which relationships between individual search states is represented. Thus, each search state contains only local information associated with the specific partial solution that the search state represents. A search state may not contain any information about other, possibly interacting search states. This is in contrast to conventional search spaces, such as the typical representation of a chess domain or an 8-puzzle domain, where each search state contains complete information about a partial solution's relationship with other paths.

To clarify this point, let a search state be defined as a set of characteristic variables. A final state is then defined by a specific set of values for some characteristic variables, or by a function of characteristic variables. In a convergent space, a specific search state may not contain all the characteristic variables necessary to determine if a final state has been reached, or to determine precisely how much progress has been made toward reaching a final state. For example, in a speech understanding domain, a search state, s, may represent a partial interpretation corresponding to an interpretation of the first 2 seconds of data. There may be another 20 seconds for which s offers no explanation. However, the problem solver cannot determine if any partial interpretations for the other 20 seconds of data exist by examining s. In contrast, in a conventional search-based chess playing program, each search state is "self-contained." The problem solver can determine whether or not each of the search states is a final state, and all of the subproblem interactions are encompassed within a search state. In such domains, two paths are never merged or combined into a single, more comprehensive partial solution.

For a given set of input data, there may be multiple instantiations of a specific terminal or nonterminal symbol resulting from multiple, non-equivalent derivation paths. This will be discussed further in subsequent chapters when the concept of *noise* and the associated ambiguity are introduced. Operators correspond to the inverse application of a production rule of the formal grammar. In addition, one of the key distinguishing characteristics of convergent search spaces is that an operator is applied to multiple states, not just one, but there is no explicit representation of the multiple states as a single state. Figure 3.2 is an example of an interpretation grammar, G', where each of the productions corresponds to a search operator[1]. Figure 3.3 represents portions of a convergent search space that corresponds to grammar G' of Fig. 3.2. In Fig. 3.3, states correspond to the elements of the grammar's alphabet, arcs between states correspond to operator applications, and the shaded areas represent those states that are only represented implicitly in a convergent search space. When an operator is applied to a state, for example, rule (or search operator) 3 is applied to state "f" to generate state "C," there is an implicit merging of states "f" and "g." The result is as if op_3 were applied to a state representing both "f" and "g." Represented graphically, it appears that search paths from "f" and "g" meet or converge at state "C."

Implicit states in a convergent search space, those shown as shaded states in Fig.3.3, are good intuitive examples of the information captured in abstract states. There is, however, a distinction between an *implicit state* and an *abstract state*. Implicit states are collections of states to which an operator appears to be applied. i.e., an implicit state is one that contains exactly

[1]In applying the IDP/*UPC* framework to more complex, real-world domains, we will use an augmented version of a grammar that makes use of the *feature list convention* discussed by Gazdar, et al.,[Gazdar *et al.*, 1982] and Knuth [Knuth, 1968]. This will increase the expressiveness of the grammar and enable it to model real world events more accurately, but it will not invalidate the analysis techniques that will be discussed later in this thesis. For the sake of clarity and simplicity, the feature list convention will not be represented in the example grammars used in this thesis unless it is explicitly stated. Our use of the feature list convention to generate problem instances is discussed further in Chapter 4.6.

Interpretation Grammar *G'*

0. S → A | B	2. B → DEW
1. A → CD	4. E → jk
3. C → fg	6. W → xyz
5. D → hi	8. j → (signal data)
7. f → (signal data)	10. k → (signal data)
9. g → (signal data)	12. x → (signal data)
11. h → (signal data)	14. y → (signal data)
13. i → (signal data)	15. z → (signal data)

Figure 3.2. Interpretation Search Operators Shown as a Set of Production Rules

the elements of the RHS of some rule of the grammar. For example, the state "f,g" or the state "x,y,z." Abstract states are not restricted to including only sets of base space states that appear in the RHS of some grammar rule. For example, the state "f,h" is an abstract state since f and h do not appear as the RHS of any rule of the grammar. Both implicit and abstract states contain information that can be exploited during problem solving.

Consideration of implicit and abstract states is also useful for understanding the concept of *potential*, which was introduced in Chapter 1 and which is formally defined in Chapter 9. In general, the potential of an operator's is a reference to the operator's ability to alter the search space in some way that reduces the cost of problem solving (or increases the effectiveness of problem solving efforts) for some other set of operators. For example, consider if there was a meta-operator that generated a state representing an abstraction of the information contained in an implicit state. In certain situations, this could be very useful information. Such a situation might be one where the problem solver was trying to extend a state h. Based on a local perspective of the problem solving situation that only included information related to state h, the problem solver would not be certain whether to choose an operator leading to an interpretation of an A or one that would lead to an interpretation of a B. However, if the problem solver could generate an abstract state that would represent whether or not an f was present, and if the problem solver could map this information back to state h, it would know exactly what to do and it could ignore alternative actions. (If an f is present, take a path leading to the generation of an A, if an f is not present, take a path leading to the generation of a B.) The degree to which this strategy reduces the cost of problem solving can be thought of as the potential of the abstract state.

Convergent search spaces are associated with a set of properties referred to as *opportunistic.* Opportunistic processing was introduced in work on the Hearsay-II project [Erman *et al.*, 1980]. Opportunistic processing enables a system to be guided by the most recently discovered results and not by a requirement to satisfy specific subgoals. In opportunistic processing systems, many redundant paths can lead to the same result and partial, intermediate results can be merged into more comprehensive results. For example, two or more subgoals could be part of the same higher-level goal and the completion of each of the tasks associated with the subgoals could independently trigger actions that would lead, redundantly, to the invocation of tasks to generate identical solutions for the higher-level goal. This can happen when a system does not fully understand the implications of a potential action. The system may not be able to determine

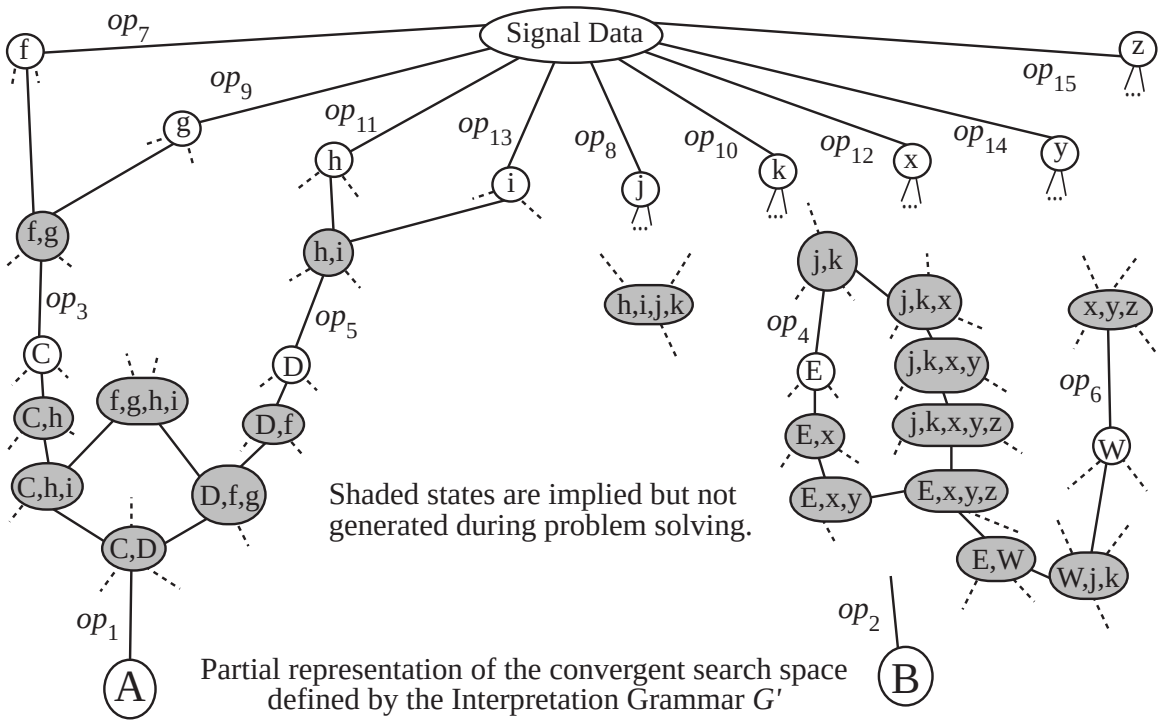


Figure 3.3. Convergent Search Space Defined by Interpretation Grammar

that two or more potential actions will generate the same result without actually invoking them. The flexibility of opportunistic processing enables a system to work around areas of a problem where the required data is of poor quality or where little constraint is generated to guide subsequent processing. After other aspects of a potential solution have been determined, they can be used to constrain search in the bypassed areas. In many ways, opportunistic processing can be thought of as an extension of bidirectional search. Opportunistic search enables problem solving to proceed bottom-up from low-level data, top-down from general expectations and abstractions, or either top-down or bottom-up from intermediate results. The advantages of opportunistic processing have been demonstrated in projects including [Erman *et al.*, 1980, Corkill, 1983, Durfee, 1987].

The interpretation search spaces presented in this thesis are all convergent search spaces. The general structures that define these spaces are defined in Chapter 3.2. More domain specific structures are defined in Chapter 4.

3.2 Defining Problem Structures

Viewing a problem solver's search operators as a formal grammar is the basis for analyzing the *component*, *cost* and *utility* [2] structures of the domain theory of an interpretation problem. Component, cost, and utility, or credibility, structures are illustrated for interpretation grammar G' in Fig. 3.4. In general, each rule of the grammar is considered to be an arbitrarily complex problem solving operator with an associated credibility and cost function. For nonterminals

[2] In the interpretation problems discussed in this thesis, utility and *credibility* are synonymous.

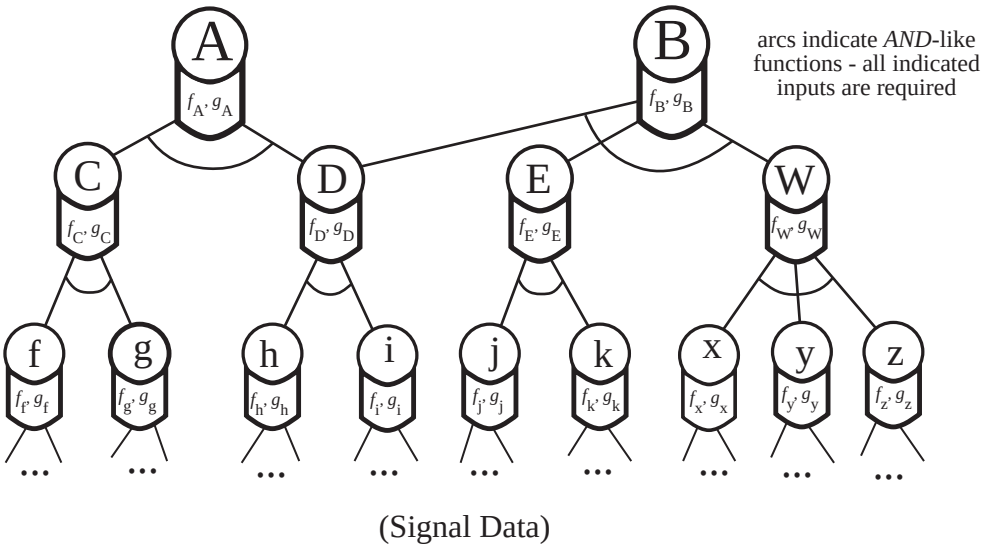


Figure 3.4. Derivation of Utility and Cost Structure From Interpretation Grammar

with multiple right hand sides (RHSs), each RHS corresponds to a unique problem solving operator. In following chapters, it will be shown that abstractions and approximations used in meta-level control actions can also be viewed from this same perspective, i.e., as operators specified by production rules of a domain's characteristic grammar.

The component structure of a problem is modeled directly by the rules of the grammar. Thus, the component subproblems of A are defined by the RHS of a rule of the grammar, $p : A \rightarrow$(component subproblems). The full structure is specified recursively. Represented graphically, the component structure of an interpretation problem appears as a derivation tree, as shown in Fig. 3.4.

The credibility structure is derived from the credibility functions corresponding to each of the production rules in the grammar. In Fig. 3.4, these functions are shown attached to the derivation tree states resulting from the inverse application of the production rule. For production rule (or state) p, the credibility function is represented f_p. Each credibility function is shown as a recursive function of a state's descendants. Thus, f_p is a function of p's descendants. In interpretation tasks, credibility functions typically include a consideration of the credibilities of the component elements and a consideration of semantics. For example, the credibility of "A" is a function of the credibilities of "C" and "D" and the results of a *semantic function*, Γ, that measures the degree to which "C" and "D" are semantically consistent. In a natural language interpretation system, a semantic function might determine the degree to which the combination of a noun phrase and a verb phrase is meaningful. If the result is nonsensical, the semantic function, Γ, will return a relatively low value. If the result is meaningful and consistent, Γ will return a relatively high value. Similarly, in an acoustic vehicle tracking system, semantic functions might check for consistent harmonics, signal energies, etc. In this thesis, we will represent credibility functions that take into consideration both the credibility and the semantic consistency of a state's descendants as, for nonterminal A with descendants C and D, $f_{\mathrm{A}}(f_{\mathrm{C}}, f_{\mathrm{D}}, \Gamma_p(\mathrm{C}, \mathrm{D}))$, where the subscript of Γ, p, is the number of the corresponding production rule from the grammar.

Similarly, the cost structure is defined in terms of the amount of a resource, such as time, required to inversely apply a production rule. For state n, the cost function is represented g_n. It is important to note that g_n defines the cost of generating the entire tree or subtree associated with state n, not just the cost of applying operator n.

This definition does not take into consideration the characteristics of the state of the problem solver at the time an operator is executed. For example, a component of an operator's cost may be a function of the amount of data currently stored in a database. Considerations such as these will not be dealt with in this thesis. However, there are a variety of techniques for dealing with them. One such technique would be to simulate an operator being applied in a series of different contexts and to compile statistical information about the cost of applying the operator and to use this information to determine the expected cost and variance of applying an operator. (In fact, this is very similar to what is done in this thesis. However, no statistical studies of operator cost were conducted. Instead, expected costs and variances were merely specified for each operator.) An alternative technique would be to incorporate a representation of significant aspects of the problem solver's internal state within each search state. This would enable our analysis techniques to differentiate search paths based on the order in which operators are applied and would enable analysis of problem solving costs with respect to the internal state of the problem solver.

In interpretation tasks, cost is typically a function of the cost of the component elements and the cost of the semantic function. For example, the cost of "A" is a function of the cost of "C" and "D" and the cost of the semantic function $\Gamma_p(\mathrm{C}, \mathrm{D})$. Cost functions are represented as, for nonterminal A with descendants C and D, $g_{\mathrm{A}}(g_{\mathrm{C}}, g_{\mathrm{D}}, cost(\Gamma_p(\mathrm{C}, \mathrm{D})))$, where the subscript of Γ, p, is the number of the corresponding production rule from the grammar and $cost(\Gamma_p(i, j, ...))$ is the cost of applying the semantic function Γ_P. Where it does not cause confusion, $cost(\Gamma_p(i, j, ...))$ will be also represented as $C(\Gamma_p(i, j, ...))$.

A typical credibility function might be "average," which determines the credibility of a state by normalizing the sum of the credibilities of the state's descendants and the result of the semantic function. Alternatives include taking the minimum value or reducing the average descendant credibility by a function of the output of the semantic function. A typical cost function might be "sum," which sums the cost of generating a state's descendants and the cost of the semantic function plus a fixed cost that represents the system overhead associated with a problem solving operator.

Interpretation Grammar *G'*

grammar rule	distribution	credibility	cost
0.1 $S \rightarrow A$	$\psi(0.1) = 0.2$	$f_{0.1}(f_A)$	$g_{0.1}(g_A)$
0.2 $S \rightarrow B$	$\psi(0.2) = 0.2$	$f_{0.2}(f_B)$	$g_{0.2}(g_B)$
0.3 $S \rightarrow M$	$\psi(0.3) = 0.2$	$f_{0.3}(f_M)$	$g_{0.3}(g_M)$
0.4 $S \rightarrow N$	$\psi(0.4) = 0.2$	$f_{0.4}(f_N)$	$g_{0.4}(g_N)$
0.5 $S \rightarrow O$	$\psi(0.5) = 0.2$	$f_{0.5}(f_O)$	$g_{0.5}(g_O)$
1. $A \rightarrow CD$	$\psi(1) = 1$	$f_1(f_C,f_D, \Gamma_1(C,D))$	$g_1(g_C,g_D,C(\Gamma_1(C,D)))$
2. $B \rightarrow DEW$	$\psi(2) = 1$	$f_2(f_D,f_E,f_W, \Gamma_2(D,E,W))$	$g_2(g_D,g_E,g_W,C(\Gamma_2(D,E,W)))$
3.0 $C \rightarrow fg$	$\psi(3.0) = 0.5$	$f_{3.0}(f_f, f_g, \Gamma_{3.0}(f,g))$	$g_{3.0}(g_f, g_g,C(\Gamma_{3.0}(f,g)))$
3.1. $C \rightarrow fgq$	$\psi(3.1) = 0.5$	$f_{3.1}(f_f,f_g,f_q, \Gamma_{3.1}(f,g,q))$	$g_{3.1}(g_f,g_g,g_q,C(\Gamma_{3.1}(f,g,q)))$
4. $E \rightarrow jk$	$\psi(4) = 1$	$f_4(f_j,f_k, \Gamma_4(j,k))$	$g_4(g_j,g_k,C(\Gamma_4(j,k)))$
5.0 $D \rightarrow hi$	$\psi(5.0) = 0.5$	$f_{5.0}(f_h,f_i, \Gamma_{5.0}(h,i))$	$g_{5.0}(g_h,g_i,C(\Gamma_{5.0}(h,i)))$
5.1. $D \rightarrow rhi$	$\psi(5.1) = 0.5$	$f_{5.1}(f_r,f_h,f_i, \Gamma_{5.1}(r,h,i))$	$g_{5.1}(g_r,g_h,g_i,C(\Gamma_{5.1}(r,h,i)))$
6.0 $W \rightarrow xyz$	$\psi(6.0) = 0.5$	$f_{6.0}(f_x,f_y,f_z, \Gamma_{6.0}(x,y,z))$	$g_{6.0}(g_x,g_y,g_z,C(\Gamma_{6.0}(x,y,z)))$
6.1. $W \rightarrow xy$	$\psi(6.1) = 0.5$	$f_{6.1}(f_x,f_y, \Gamma_{6.1}(x,y))$	$g_{6.1}(g_x,g_y,C(\Gamma_{6.1}(x,y)))$
7. $f \rightarrow (s)$	$\psi(7) = 1$	$f_7(f_{(s)}, \Gamma_7(\ (s)\ ))$	$g_7(g_{(s)},C(\Gamma_7(\ (s)\ )))$
8. $j \rightarrow (s)$	$\psi(8) = 1$	$f_8(f_{(s)}, \Gamma_8(\ (s)\ ))$	$g_8(g_{(s)},C(\Gamma_8(\ (s)\ )))$
9. $g \rightarrow (s)$	$\psi(9) = 1$	$f_9(f_{(s)}, \Gamma_9(\ (s)\ ))$	$g_9(g_{(s)},C(\Gamma_9(\ (s)\ )))$
10. $k \rightarrow (s)$	$\psi(10) = 1$	$f_{10}(f_{(s)}, \Gamma_{10}(\ (s)\ ))$	$g_{10}(g_{(s)},C(\Gamma_{10}(\ (s)\ )))$
11. $h \rightarrow (s)$	$\psi(11) = 1$	$f_{11}(f_{(s)}, \Gamma_{11}(\ (s)\ ))$	$g_{11}(g_{(s)},C(\Gamma_{11}(\ (s)\ )))$
12. $x \rightarrow (s)$	$\psi(12) = 1$	$f_{12}(f_{(s)}, \Gamma_{12}(\ (s)\ ))$	$g_{12}(g_{(s)},C(\Gamma_{12}(\ (s)\ )))$
13. $i \rightarrow (s)$	$\psi(13) = 1$	$f_{13}(f_{(s)}, \Gamma_{13}(\ (s)\ ))$	$g_{13}(g_{(s)},C(\Gamma_{13}(\ (s)\ )))$
14. $y \rightarrow (s)$	$\psi(14) = 1$	$f_{14}(f_{(s)}, \Gamma_{14}(\ (s)\ ))$	$g_{14}(g_{(s)},C(\Gamma_{14}(\ (s)\ )))$
15. $z \rightarrow (s)$	$\psi(15) = 1$	$f_{15}(f_{(s)}, \Gamma_{15}(\ (s)\ ))$	$g_{15}(g_{(s)},C(\Gamma_{15}(\ (s)\ )))$
16. $M \rightarrow Y$	$\psi(16) = 1$	$f_{16}(f_Y)$	$g_{16}(g_Y)$
17.0 $Y \rightarrow qr$	$\psi(17.0) = 0.5$	$f_{17.0}(f_q,f_r, \Gamma_{17.0}(q,r))$	$g_{17.0}(g_q,g_r,C(\Gamma_{17.0}(q,r)))$
17.1 $Y \rightarrow qhri$	$\psi(17.1) = 0.5$	$f_{17.1}(f_q,f_h,f_r,f_i, \Gamma_{17.1}(q,h,r,i))$	$g_{17.1}(g_q,g_h,g_r,g_i,C(\Gamma_{17.1}(q,h,r,i)))$
18. $N \rightarrow Z$	$\psi(18) = 1$	$f_{18}(f_Z)$	$g_{18}(g_Z)$
19. $Z \rightarrow xy$	$\psi(19) = 1$	$f_{19}(f_x,f_y, \Gamma_{19}(x,y))$	$g_{19}(g_x,g_y,C(\Gamma_{19}(x,y)))$
20. $O \rightarrow X$	$\psi(20) = 1$	$f_{20}(f_X)$	$g_{20}(g_X)$
21.0. $X \rightarrow fgh$	$\psi(21.0) = 0.5$	$f_{21.0}(f_f,f_g,f_h, \Gamma_{21.0}(f,g,h))$	$g_{21.0}(g_f,g_g,g_h,C(\Gamma_{21.0}(f,g,h)))$
21.1. $X \rightarrow fg$	$\psi(21.1) = 0.5$	$f_{21.1}(f_f,f_g, \Gamma_{21.1}(f,g))$	$g_{21.1}(g_f,g_g,C(\Gamma_{21.1}(f,g)))$

(s) = signal data $\Gamma_n(i,j,...)$ = semantic evaluation function for rule n $C(\Gamma_n(i,j,...))$ = cost of executing $\Gamma_n(i,j,...)$

Figure 3.5. Example of Interpretation Grammar with Fully Specified Distribution, Credibility, and Cost Functions

For a given IDP instance, I, the problem structure of the domain will now be defined in terms of G_I, the characteristic grammar for I's domain, and the functions f_p and g_p. Formally,

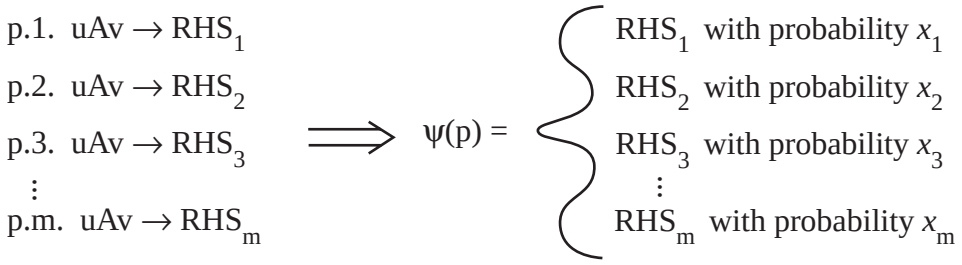
Definition 3.2.1 *An IDP Grammar is a grammar, $G_I =< V, N, SNT, S, P >$, where V is the set of terminal symbols, N is the set of nonterminal symbols, SNT is the set of* solution-nonterminal *symbols that correspond to final states, S is the start symbol for the grammar*[3]*, and P is the set of context-free production rules.*

In the analysis framework, an IDP problem will be represented with two distinct grammars, the generation grammar, IDP$_g$, and the interpretation grammar, IDP$_i$. IDP$_g$ is used to model the structure of a domain and the domain events that resulted in the creation of the observed signal data. Thus, IDP$_g$ will be shown with the characteristics of G_I that are applicable to generation, the grammar rules associated with the problem domain and their distribution functions. IDP$_i$ is used to model a problem solver's architecture and will be shown with cost and semantic functions. When a new IDP grammar is introduced, it will be shown with both IDP$_g$ and IDP$_i$ in a combined representation including cost, credibility, and distribution functions. As described in Chapter 7, IDP$_g$ and IDP$_i$ can differ significantly and their respective sets of production rules and non-terminals can be quite different, even disjoint. Even so, we will often refer to both of them as "an IDP grammar."

The G_I grammar specification is very similar to traditional grammar specifications with the notable exception of the set of solution nonterminals. This specification is necessary for analytical reasons that will be explained in Chapter 5.1.1. f_p and g_p are defined for elements $p \in P$. In addition, another function, ψ, will be used to define problem structures by augmenting G_I's expressive power. Specifically, for each rule $p \in P$, there is a corresponding $\psi(p)$. $\psi(p)$ will be used to represent the distribution of "right hand sides (RHSs)" associated with p. Each of these alternative RHSs will be thought of as a distinct problem solving operator that is associated with a distinct credibility and cost function. This is illustrated in Fig. 3.5. ψ is introduced here to support the definition of problem structures. ψ will be used to support the modeling of real-world phenomena such as uncertainty caused by noise, missing data, distortion and masking (see Chapter 4). ψ will be represented by saying that, for production p which decomposes to RHS$_1$, there are one or more corresponding p' that decompose to alternative RHSs, RHS$_2$ ... RHS$_m$. The definition of ψ is shown in Fig. 3.6. In Fig. 3.6, production rule p has a number of possible RHSs. These different RHSs can be thought of as the right hand sides of variations of p numbered p.1 through p.m and, for a given p, $\sum_n \psi(p.n) = 1$. Figure 3.5 shows an example of a simple interpretation grammar with fully specified distribution, credibility, and cost functions. For some of the rules, e.g., 0.1, 0.2, 20, etc., there are no semantic functions. This is reflected in the credibility and cost functions that only take into consideration the credibility and cost of descendant states.

In this thesis, the ψ values that are used in the examples and the experiments are all *exact* values. In other words, all ψ values correspond to precise ratios and do not correspond to expected values and variances. This results in a very precise predictive capability, as shown in the experimental results in Chapters 7, 11, and 13. Future research projects will involve supplementing the representation to include a form of uncertainty where the values of ψ are

[3] Note that in the state space derived from an IDP specification, no search states are created that correspond to the start symbol, S.



Given an interpretation rule, p, with m semantically equivalent RHSs, the function ψ(p) specifies the distribution of the RHSs.

Figure 3.6. Example of the Distribution Function ψ

not known with such precision but where they are represented with an expected value and a variance.

In general, there are no restrictions on the production rules of G_I. In practice, however, it will be necessary to limit any recursive rules to a specific number of iterations in order to perform numerical analyses. This will not limit the applicability of the analysis tools in any way. This is because real-world interpretation systems must function with similar restrictions in that each problem solving instance must be of finite length. For systems that are intended to interpret streams of continuous data, the data is divided into "time slices" of finite length. In these systems, our analysis techniques would be applied only to the individual time slices. The techniques described in this thesis will have to be extended to address issues associated with problem solving systems that process data from multiple time slices simultaneously.

It is important to note that multiple final states may correspond to each element of SNT. For example, in a natural language domain, SNT may contain a single element, "sentence." In a typical problem solving instance, there will be multiple (often numerous) different sentences generated. Similarly, in a vehicle tracking domain, a typical element of SNT might be "vehicle track of type 1" and any given problem solving instance may generate multiple, different interpretations of type 1 vehicle tracks. The alternative instantiations of an SNT are differentiated by the characteristics of the specific instances, i.e., their characteristic variables. For example, the alternative type 1 vehicle tracks may pass through different locations.

3.3 Structural Interaction

The preceding section defined the component, credibility, and cost structures of a domain as independent entities. In fact, in many domains, these structures interact. For example, the cost of a semantic function may depend on the dynamically computed probability associated with a search path. If the problem solver determines that the probability of reaching a final state is high, it may decide to execute a more costly semantic function in order to generate the best possible answer. Alternatively, if the problem solver determines that there is little chance a search path will reach a final solution, it may devote little effort to a semantic function. Such interactions can lead to very complex problem solving behaviors and to corresponding analysis techniques that are also very complex. In the examples in this thesis, analysis techniques that are appropriate for various structural interactions are presented and discussed. However, we will

often use approximations of these techniques for computability reasons. Chapter 6.7 presents formal definitions and examples of structural interactions.

3.4 Interpretation Problem Solving and Formal Problem Solving Paradigms

As discussed by Kanal and Kumar in [Kumar and Kanal, 1988], formal problem solving paradigms that are based on search techniques can be divided into two general categories, either *top-down* or *bottom-up*. The IDP/*UPC* framework is consistent with this and supports the analysis of control architectures from either a top-down or a bottom-up perspective. This is important because it is hoped that the IDP/*UPC* framework can be used to develop a unified perspective of problem solving by extending Kanal and Kumar's taxonomy to include the sophisticated control architectures that can be described with the IDP/*UPC* framework.

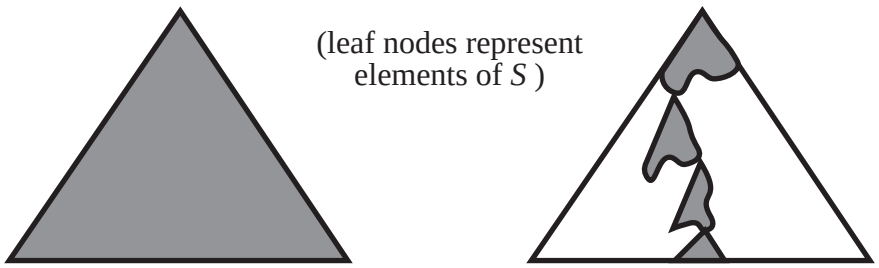
Using bottom-up control architectures, rules of the grammar are inversely applied and more comprehensive derivation trees are generated and represented as new states. Thus, larger problems are solved starting with their smaller components. This approach is used in many search procedures and is the basis for dynamic programming [Papadimitriou and Steiglitz, 1982, Pearl, 1984, Kumar and Kanal, 1988]. The examples in Chapter 3.1 are representative of bottom-up problem solving in interpretation problems.

Using top-down control architectures, some representation of the total set of interpretations is repeatedly partitioned and pruned. After each partition, all members of the partition are deleted for which it can be shown that, even after elimination, the most credible interpretation is an element of one of the remaining partitions. This technique has been used extensively in Operations Research, where it is referred to as Branch-and-Bound [Papadimitriou and Steiglitz, 1982, Pearl, 1984, Kumar and Kanal, 1988]. Top-down and bottom-up architectures have been effectively combined in a number of systems such as blackboard systems [Erman *et al.*, 1980, Corkill, 1983]. In these systems, the top-down strategies, such as goal processing techniques, can be considered meta-control architectures because they use abstract or approximate states.

Using either or both of these approaches, problem solving continues until *every possible* derivation tree is either generated or eliminated from consideration based on the structure of the problem. Once all derivation trees have been determined, the problem solver identifies which has the highest rating and returns that derivation tree as the interpretation.

This definition is important because it implies that the problem solver cannot simply compute one derivation tree and stop. In order to find the "best" interpretation, *every* possible search path must be explored in some way. Thus, for any given state, at termination, every operator that can be applied to it has been accounted for in some way. Such a state will be referred to as a *connected state* and a space consisting solely of connected states will be referred to as a *connected space*. An *open state* will be a state that is not connected. The set I of interpretations from Definition 3.1 corresponds to a connected space. Implicit in this definition is the requirement that a connected space "account for" or "explain" all the input data. This requirement is derived from the requirement that each individual solution that is considered as an overall solution to a specific problem must explain all the detected input data. The distinction of "detected data" is important because the analysis techniques only work for the problem solving actions represented as search processes. They do not necessarily explain all the data input to the non-search processes such as low-level data filtering operators.

Clearly, depending on a domain's characteristic grammar and associated functions, interpretation problem solving can be a formidable task. The total number of solutions generated for an arbitrary set I can be enormous. In general, the role of sophisticated control is to



a. Without the use of sophisticated control, search spaces must be enumerated exhaustively.

b. Using sophisticated control, many areas of a search space can be eliminated from consideration implicitly, thereby reducing the cost of problem solving.

Figure 3.7. Implicit Enumeration – the Role of Control

limit the size of I by implicitly enumerating as much of the search space as possible. This objective is illustrated in Fig. 3.7, from Berliner [Berliner, 1979]. Figure 3.7.a represents the set I generated by grammar G without the use of sophisticated control techniques. In this example, problem solving is essentially exhaustive – every possible derivation tree is generated and compared. Figure 3.7.b shows the effects of sophisticated control. Here, large portions of the search space are eliminated from consideration based on the problem solver's understanding of the problem's structure.

Finally, it is important to emphasize that G_I, f_p, g_p, and ψ are used to specify the set I from which the best interpretation will be identified.

3.5 Chapter Summary

This chapter defines the specific class of problems, the *interpretation decision problem (IDP)*, studied in this thesis. An IDP is a constructive search problem where, given an input string of signal data, the problem solver tries to find the *best* interpretation corresponding to the events that could have caused the signal data. IDP problems are defined in terms of discrete optimization problems. Specifically, the problem solver is trying to determine which interpretation from a set of possible interpretations has the highest credibility. This defines the problem in such a way that it is possible to quantitatively analyze it. This is because it provides a clear criterion for termination – problem solving halts when every possible search path has been connected to a final state or eliminated from consideration. At this point the problem solver conducts a linear search of the set of interpretations that were found to determine which has the highest credibility. Also, since it casts interpretation as a discrete optimization problem, it clearly defines interpretation's relationship to other classes of problems and it makes it possible to understand the general applicability of algorithms and other results that are constructed determined for interpretation domains.

In IDP models, the different feature structures that are defined by domain theories are combined into a unified representation by expressing them in terms of formal grammars and functions associated with production rules of the grammar. Nonterminals of the grammar represent intermediate problem solving states, terminal symbols represent raw sensor input, and

the production rules of the grammar represent potential problem solving actions. The grammar rules of IDP models specify the component structure of a domain and each production, p, has associated cost and utility functions, g_p and f_p, that define the cost and utility structures. In addition, IDP models explicitly represent aspects of inherent uncertainty in a domain with the distribution function, ψ, that defines the probability structure of the domain. (i.e., ψ, along with other mechanisms, define inherent uncertainty in a domain.) For a given production, p, the frequency of the occurrence of p's right-hand-side (RHS) is specified by the distribution function $\psi(p)$. Thus, p can have multiple RHSs, RHS_1 through RHS_n, and the distribution of the RHSs is defined by $\psi(p)$. Each production rule, p, is associated with a semantic function, Γ_p that is a function of the subtree components represented by the elements on the right-hand-side of p. Γ_p measures the "consistency" of the semantics of its input data and returns a value that is included in the credibility function. For example, in a speech understanding domain, Γ_p would rate the consistency of the meaning of a sentence and return a value indicating whether or not the sentence made any sense. An IDP grammar also includes an extension to its representation called a *feature list* to represent the characteristics of real-world phenomena.

CHAPTER 4

DEFINING IDP STRUCTURES

Chapter 3 introduced the IDP formalism for modeling domain theory problem structures. The significance of these structures is that they are defined in terms of the operators available to solve the problem. Conversely, problem solving operators are defined in terms of the structure of a problem's domain theory.

This chapter will demonstrate how the IDP formalism can define a number of important domain theory problem structures. As noted above, these structures also define the operators and control actions available to solve a problem. Chapters 6 and 8 define another formalism, the *UPC* formalism, that will be used in conjunction with the IDP formalism as a general model of control and problem solving activities. Using the *UPC* formalism, the cost and utility of control and problem solving actions will be compared directly. The chapters describing the *UPC* model will draw heavily on the structures defined in this chapter.

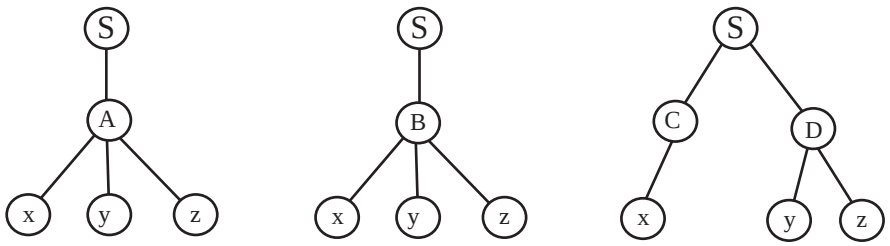
The power of the IDP formalism lies in its ability to model the structure of a problem domain. The key to exploiting the IDP formalism is based on the degree to which the structure of a problem formulation maps into and is represented by the IDP's grammar, credibility functions, cost functions, and RHS distribution functions.

For this thesis, analysis will focus on five primary aspects of problem structure: uncertainty, operator organization, redundancy, interacting subproblems, and bounding functions. These aspects of the structure of interpretation problems are introduced in this section. Two of these structural aspects, uncertainty and redundancy, are inherent components of a domain. They are derived naturally by representing a problem domain as an IDP model. The other aspects can be thought of as transformations of the base IDP model. These transformations are used to model control actions in terms of a unified perspective with problem solving actions. Typically, these transformations are used to exploit structures present in the base IDP model.

Undoubtedly, many other structures are manifested in a given problem domain. However, it is far beyond the scope of this work to attempt to classify, or even identify all possible aspects of problem space structure.

4.1 Inherent Uncertainty

One of the more difficult aspects of an interpretation task is dealing effectively with *inherent uncertainty*. In terms of the IDP formalism, inherent uncertainty can be thought of as ambiguity in the interpretation grammar. Ambiguity increases the complexity of an interpretation problem by making a large number of partial solutions and full, potential solutions seem plausible. This implies that additional operator applications must be used to generate and differentiate the ambiguous interpretations, as will be discussed in the next section. Thus, inherent uncertainty structures are not used to model control actions, rather, they are structured phenomena that control actions are intended to exploit to improve the efficiency of problem solving. Using the IDP model, inherent uncertainty will be classified into the taxonomy presented in the following



a. An ambiguous grammar. There are three parse trees for the string xyz.

Figure 4.1. An Example of Ambiguity

sections. The definitions of the classes of inherent uncertainty will be associated with the rules of the grammar and with a formal definition of ambiguity.

For a rule of the grammar, p, variations of the rule will be used to help define certain forms of uncertainty. This will be represented by associating multiple RHSs to p. The distribution of the domain events that determine which of the RHSs is appropriate will be modeled by $\psi(p)$. This is an important point that will be emphasized in Chapter 6. Finally, each of the RHSs will be thought of as a unique primitive operator that the problem solver can use to build a partial or full interpretation.

The formal definition of ambiguity that will be used to define the taxonomy of inherent uncertainty structures is presented in the next section.

4.1.1 Ambiguity

Ambiguity is a property of a grammar that leads to multiple interpretations for a given input. In an ambiguous IDP, the only way to differentiate a correct interpretation from an incorrect interpretation is to compare their credibility values. More formally:

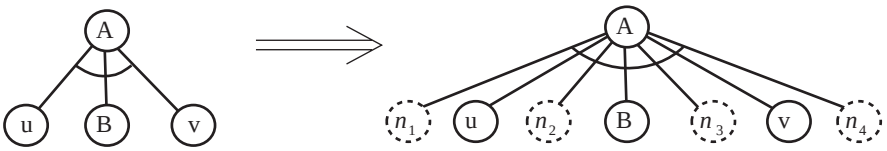
Definition 4.1.1 *Ambiguity - An instance of an IDP is* ambiguous *iff two (or more) interpretation trees exist for some input, X. (A more precise definition of ambiguity will be given in Definition 6.3.14.)*

An example of an ambiguous grammar is shown in Fig. 4.1 where three interpretation trees exist for the terminal string xyz.

IDPs can also be *semi-ambiguous.*

Definition 4.1.2 *Semi-Ambiguity - An instance of an IDP is semi-ambiguous iff the IDP is not ambiguous and two (or more) interpretation subtrees exist for some portion, Y, of an input string, X.*

In a semi-ambiguous IDP, it is not possible to differentiate alternative interpretation subtrees, $(y_1, \ldots y_n)$, that are components of the best interpretation ("correct" subtrees) from subtrees that are not components of the best interpretation ("incorrect" subtrees) simply by comparing their credibility levels and choosing the subtree with the higher rating. This is because IDP domains are non-monotone. (Non-monotonicity is formally defined in Chapter 4.5.) The only way to determine which of the subtrees, $(y_1, \ldots y_n)$, is the correct subtree is to continue the interpretation, extending each of the competing alternatives. One of two things will happen, either the problem solver will be unable to extend a partial interpretation



b. The production rule p: A -> uBv and the corresponding `noisey' rule p': A -> $n_1un_2Bn_3vn_4$.

Figure 4.2. An Example of a Noisy Grammar Rule

or the problem solver will find a solution. If the problem solver fails to extend all partial interpretations derived from a given subtree, then that subtree is incorrect. When the problem solver identifies the best interpretation, all the subtrees associated with that interpretation are considered correct partial interpretations.[1]

Intuitively, the cost of ambiguous IDPs is the sum of the costs of generating each of the derivation trees corresponding to ambiguous interpretations and the subtrees, $(y_1, \ldots y_n)$, corresponding to partial interpretations, plus the cost of differentiating the correct interpretation (and, by implication, the correct derivation subtrees). In general, as the number of ambiguous interpretations and partial interpretations increases in size, the cost of problem solving will increase.

4.1.2 Noise

Intuitively, noise can be thought of as domain data that is generated in a spurious fashion and that is not necessary or sufficient to infer the associated higher-level partial or full interpretation. Noise does not necessarily correspond exclusively to unknown phenomena. For example, in a vehicle tracking domain, noise may correspond to relatively normal domain events, such as sounds associated with weather phenomena or animals. Formally,

Definition 4.1.3 *Noise - A grammar, G, is subject to noise iff*
$\exists$ *a rule p* $\in$ *P, where p:* $A \rightarrow uBv \Rightarrow p'$*:* $A \rightarrow n_1 u n_2 B n_3 v n_4$*, for u,v* $\in$ *(V* $\cup$ *N)*, A* $\in$ *(SNT* $\cup$ *N), B* $\in$ *(V* $\cup$ *N),* $n_i \in (V \cup N)^*$*, and* $f_p(f_u, f_B, f_v, \Gamma_p(u, B, v)) > f_{p'}(f_{n_1}, f_u, f_{n_2}, f_B, f_{n_3}, f_v, f_{n_4}, \Gamma_{p'}(f_{n_1}, u, f_{n_2}, B, f_{n_3}, v, f_{n_4}))$ *for maximum ratings of the* f_{n_i}*. The distribution of* p *and* p' *is modeled by* ψ*.*

In other words, for some rule p in G, there are at least two possible RHSs. The frequency with which noise appears in a domain is modeled by ψ. Figure 4.2 is a representation of the definition of noise. Each of the states labeled n_i could lead to arbitrary "noisy" subtrees[2]. In many domains, each production rule may have numerous alternative noisy productions. Again, the determination of which of these productions is used is made by the function ψ. It

[1] It is important to note that virtually all interpretation problems are at least semi-ambiguous. An unambiguous interpretation problem might be more properly categorized as a parsing problem or a classification problem.

[2] Please note that in the figures in this thesis, and in the IDP/*UPC* Framework, there is no meaning associated with the order in which terminal or nonterminal symbols are represented. Thus, the sequence of terminal symbols "r h i" is identical to the sequence "h r i."

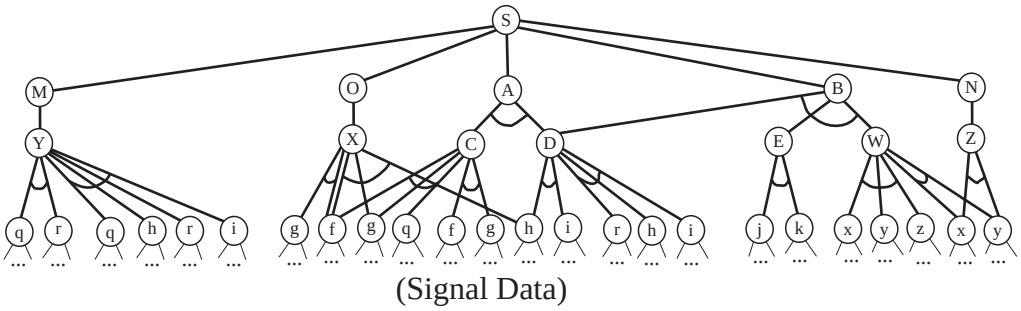


Figure 4.3. Interpretation Grammar G' with Added Noise and Missing Data Rules

is important to emphasize that, by definition, a noisy rule has a credibility that is lower than the corresponding non-noisy rule. Given a set of RHSs that are candidates for being returned by ψ, the non-noisy RHS is the one with the highest credibility. (In situations where a noisy rule has higher credibility than a non-noisy rule, the noisy rule would be considered to be the "correct" rule and the non-noisy rule to be a missing data rule. Missing data rules are defined below.)

Noise is problematic for an interpretation problem because it can increase the amount of ambiguity with which the problem solver must contend. This increases the cost of problem solving. An increase in ambiguity implies that the problem solver must apply additional operators, which increases cost, to build and differentiate alternative interpretations. Consider a grammar in which no two rules have identical right hand sides (RHSs). (Note that this IDP may or may not be ambiguous.) Now assume that noise is introduced in such a way that at least one of the "noisy" rules has an RHS that is identical to the RHS of an existing, non-noisy rule. An example of such a situation is shown by Figs. 4.3 and 4.4. In Fig. 4.3, the signal input "f g q r h i" can only be associated with the derivation of a "C" and a "D" and, subsequently, and "A." Given the additions to the grammar shown in Fig. 4.4, the signal input "f g q r h i" can lead to the interpretations of either an "M" or an "A." This is because "f g h i" is used to derive "noise" and "f r" is used to derive "Y." "Y" and "noise" are used to derive an "M." By definition, the IDP represented by these figures is at least semi-ambiguous.

Given a problem instance in which the noisy rule is the correct interpretation for the signal data, the problem solver will have to differentiate the interpretation in which the noisy rule is used from interpretations corresponding to "non-noisy" interpretations of the data. This can be very difficult because the only way to differentiate a correct interpretation tree from an incorrect one is by comparing their credibility ratings and, as was discussed previously, this technique does not work, in general, for partial trees. From the local perspective of competing alternative partial interpretation trees, relative credibility ratings may correlate to the probability of a given subtree being correct. However, this correlation may be significantly less than one. Noisy production rules often have low credibility ratings and correct subtrees that contain noisy production rules frequently have lower credibility ratings than incorrect alternative subtrees. Consequently, the problem solver may not be able to differentiate competing alternative interpretations without additional, perhaps expensive, processing. It may be the case that, in order to differentiate competing alternatives in a semi-ambiguous grammar, the problem solver is forced to generate

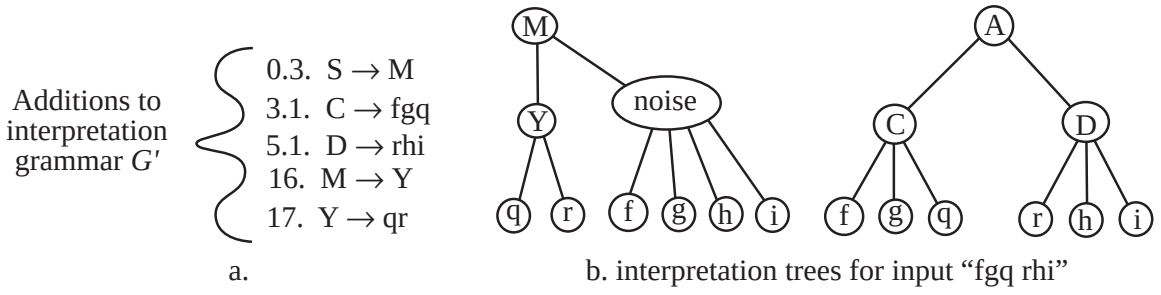


Figure 4.4. An Example of Correlated Noise - The noise in rules 3.1 and 5.1, q and r, is correlated to an interpretation of M.

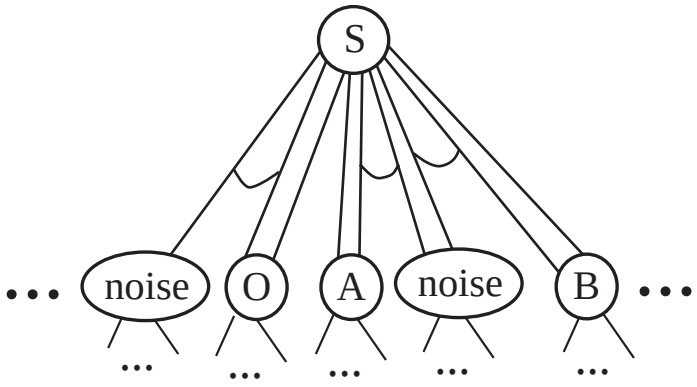


Figure 4.5. Implicit Rules for Interpreting Noise

nearly complete solutions incorporating each of the competing alternatives. The need to apply additional operators to build and differentiate alternative subtrees could result in very expensive problem solving.

In Fig. 4.3, noise has been added to grammar *G′* from Fig. 3.2. In this grammar, SNT contains A, B, M, N, and O. Some of the noise is specified in the new rules 3.1 and 5.1 shown in Fig. 4.4. Also, a new interpretation, M (rule 16), has been added to the grammar. (Note that each of these rules corresponds to the addition of a new primitive operator.) As a result of adding these production rules, in certain noisy situations, signal data can be interpreted as either an "M" or an "A." Figure 4.4, which shows alternative derivation trees for "M" and "A" given the input "fgq rhi," depicts such a situation.

From a local perspective, the partial interpretation corresponding to state Y may be rated higher than either of the partial interpretations corresponding to states C or D. However, the full interpretation represented by state M will have a lower rating than the full interpretation represented by state A. Intuitively, this is a result of the fact that the A interpretation explains all the observed phenomena and the M interpretation only explains the data related to q and r. To account for data that is not explained by an interpretation, we assume the existence of rules such as those in Fig. 4.5 that account for extraneous noise data. In a typical situation, an interpretation that includes noise as shown in this figure has a much lower credibility than an interpretation that explains all of the data as being something other than random noise.

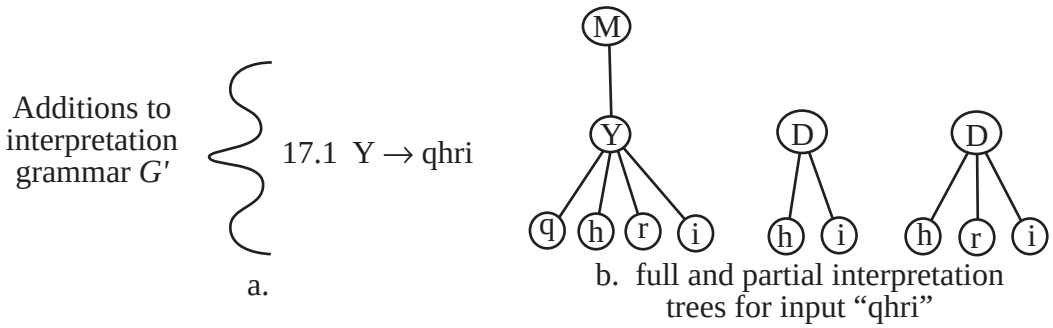


Figure 4.6. An Example of Uncorrelated Noise

This is an example of how noise can add ambiguity to a grammar. Without the noise, when the problem solver recognizes a q or an r, it can immediately return an interpretation of M, since no other interpretation contains a q or an r. However, with the addition of noise, the problem solver must differentiate interpretations of M and A. This will be much more expensive because the problem solver will have to apply all the operators implied by each interpretation's component structure in order to generate accurate ratings. This is necessary because the problem solver will have to execute all the semantic functions to generate the credibility ratings needed to differentiate the alternative interpretations.

Figure 4.6 illustrates a slightly different situation that is also based on the modifications to *G′* shown in Fig. 4.3. In this example, noise is added to grammar *G′* in the form of rule 17.1, and this noise leads to the generation of the input "qhri." During interpretation, two partial interpretations corresponding to D are generated. However, because there is no additional data, interpretations involving the use of D fail to generate an A interpretation and the D states become connected without being used in a full interpretation. This is an example of how noise can make a grammar semi-ambiguous. Though the increase in cost may not be as severe as an ambiguous case, semi-ambiguity still increases interpretation costs because it forces the problem solver to determine which alternative partial interpretations cannot be extended to full interpretations.

These examples, and the observation that noise may lead to rules with identical RHSs and ambiguity, lead to definitions for two specific kinds of noise.

Definition 4.1.4 *Correlated Noise - Noise that results in ambiguous interpretations. Correlated noise can significantly increase the amount of work or other resources required to solve an IDP if it requires the problem solver to generate multiple interpretations in order to differentiate the competing partial interpretations. Correlated noise is especially problematic when the correlation is high and the introduction of noise leads to numerous incorrect, but credible, interpretations. An example of correlated noise is shown in Fig. 4.4.*

Definition 4.1.5 *Uncorrelated Noise - Noise that does not result in ambiguous interpretations. Uncorrelated noise may lead to semi-ambiguous IDPs, but, in general, is easier to differentiate than correlated noise. With uncorrelated noise, the problem solver is not required to generate complete interpretation trees to differentiate the correct noisy partial interpretation from incorrect competing alternatives. Instead, the competing alternatives will be eliminated when the problem solver fails*

to extend them at some point of the interpretation process. The incorrect interpretations will not correspond to any possible interpretation and the problem solver will not be able to inversely apply any production rules to extend the interpretation. An example of uncorrelated noise is shown in Fig. 4.6. Also, as will be discussed, additional problem structures defined by bounding functions *(see Chapter 4.5), which represent pruning functions, can be used to identify incorrect partial interpretations without the derivation of a full interpretation.*

The correlation of noise to potentially correct interpretations is much more complex than the dichotomy suggested by the above definitions. In fact, the correlation of noisy interpretations to correct interpretations can be thought of as a continuum in terms of cost. At one extreme is noise that is uncorrelated to correct data. This noise can be disambiguated with little cost. At the other extreme is noise that leads to numerous full interpretations and significant cost. Between these extremes are cases where the noise is uncorrelated to actual data, but where the structural constraints of the problem that enable incorrect, uncorrelated partial interpretations to be pruned are costly to apply. For example, the problem solver may expend a great deal of effort on a highly rated partial interpretation before encountering grammar constraints that prevent its further extension. Thus, the concept of the correlation of noise to potentially correct, alternative interpretations will be thought of as a continuum where low-correlation implies that incorrect interpretations resulting from noise can be disambiguated with little cost and high-correlation implies that incorrect interpretations resulting from noise will be very expensive to disambiguate.

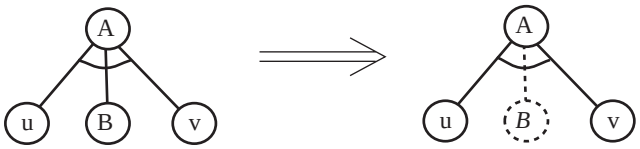
4.1.3 Missing Data

Intuitively, missing data is the result of sensing and other domain phenomena that prevent certain data normally associated with a domain event from being detected. For example, in a vehicle monitoring domain, the microphones used to pick up signals from vehicles might be flawed so that some frequencies are missed in situations where the vehicle is moving slowly. The characteristic signals are present, but are below some threshold and are not 'heard' by the sensors. Missing data also results from inappropriately processed low-level data [Lesser *et al.*, 1993]. For example, a filtering algorithm may be used with a poor selection of tuning parameters resulting in the loss of significant data. Formally,

Definition 4.1.6 *Missing Data - A grammar, G, is subject to missing data iff* $\exists\ p \in P$, *where p:* $A \rightarrow uBv \Rightarrow p'$: $A \rightarrow uv$, *for u,v* $\in (V \cup N)^*$, $A \in (SNT \cup N)$, $B \in (V \cup N)$,*and* $f_p(f_u, f_B, f_v, \Gamma_p(u, B, v)) > f_{p'}(f_u, f_v, \Gamma_{p'}(u, v))$. *In addition, the distribution of* p *and* p' *is modeled by* ψ.

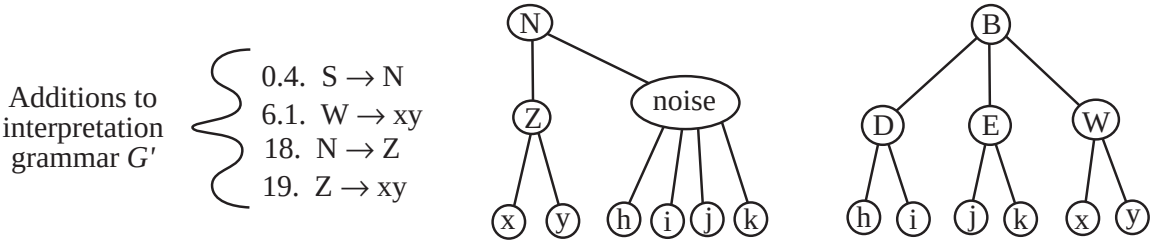
As with noise, missing data implies the existence of multiple RHSs for some production rule(s). The distribution of the domain events that determine which of the RHSs is appropriate is modeled by ψ. Figure 4.7 illustrates the definition of missing data. In many domains, each production may correspond to numerous alternative missing data productions. Again, by definition, a missing data rule has a credibility that is lower than the corresponding "complete-data" rule. Missing data can increase the ambiguity of an IDP and, in most cases, this will increase the cost of differentiating correct interpretations from incorrect interpretations.

Figure 4.3 also shows G' with added rules representing missing data rules. Figure 4.8 shows an example of a situation where missing data leads to multiple interpretations of signal



c. The production rule p: A -> uBv and the corresponding `missing data' rule p': A -> uv.

Figure 4.7. An Example of a Missing Data Grammar Rule

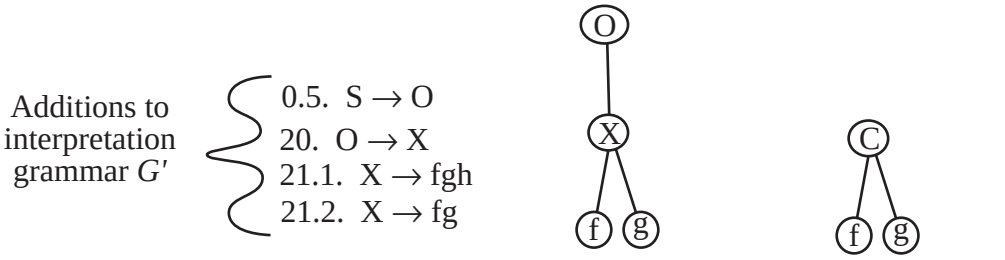


a. b. interpretation trees for input "hi jk xy"

Figure 4.8. An Example of Correlated Missing Data - The data missing in rule 6.1, w, is correlated to an interpretation of N.

data. In this example, missing data is added to grammar G' in the form of rule 6.1. The signal input to the problem solver is "hi jk xy" and, as a result of missing data rule 6.1, the data can be interpreted as a B. However, the same data can also be interpreted as an N. The determination of which is correct cannot be made without additional processing. The partial interpretation corresponding to state Z may have a higher rating than the partial interpretations corresponding to states D, E, or W. The differentiation must be made between states N and B. Here, B will have a higher rating since it explains all the signal data and N only explains the data corresponding to x and y. This is an example of missing data causing ambiguity.

Figure 4.9 depicts a situation where missing data again leads to multiple interpretations. This situation is also based on the grammar shown in Fig. 4.3. In this example, missing data is added to grammar G' in the form of rule 21.2. In this situation, an interpretation of O



a. b. full and partial interpretation trees for input "fg"

Figure 4.9. An Example of Uncorrelated Missing Data

will result from the input "fg." In addition, partial interpretation C will be formed. Partial interpretation C will not be extended since the data needed to generate an A is not present. Therefore, C will be connected without resulting in the generation of a full interpretation. This is an example of missing data causing semi-ambiguity.

Missing data can also be categorized into two broad classes.

Definition 4.1.7 *Correlated Missing Data – Missing data that results in ambiguous interpretations. Similar to correlated noise, correlated missing data can significantly increase the amount of work or other resources required to solve an IDP if it requires the problem solver to generate multiple interpretations in order to differentiate the competing partial interpretations. Correlated missing data is especially problematic when the correlation is high and the introduction of missing data leads to numerous incorrect, but credible, interpretations. Figure 4.8 shows an example of correlated missing data.*

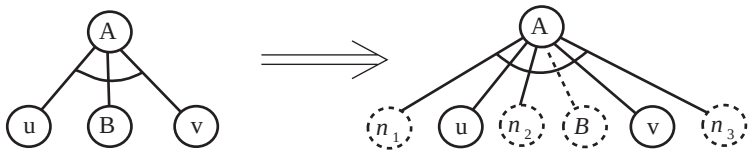
Definition 4.1.8 *Uncorrelated Missing Data – Missing data that does not result in ambiguous interpretations. Similar to uncorrelated noise, uncorrelated missing data may lead to semi-ambiguous IDPs, but, in general, is easier to differentiate than correlated missing data. With uncorrelated missing data, the problem solver is not required to generate complete interpretation trees to differentiate the correct missing data partial interpretation from incorrect competing alternatives. Instead, the competing alternatives will be eliminated when the problem solver fails to extend them at some point of the interpretation process, either as a result of syntactic constraints of the grammar or boundary function constraints (see Chapter 4.5) defined by the functions* f_p *and* Γ_p *. In other words, the incorrect partial interpretations will not correspond to any possible full interpretation and the problem solver will not be able to inversely apply any production rules to extend the interpretation. Figure 4.9 shows an example of uncorrelated missing data.*

As with noise, the degree to which missing data is correlated to other domain events is a continuum from uncorrelated to correlated. Again, the continuum is expressed in terms of the additional cost that must be incurred before enough constraints are applied to eliminate an incorrect partial interpretation from further consideration. In other words, highly correlated missing data will increase interpretation cost significantly and missing data with low correlation will increase interpretation cost only slightly.

4.1.4 Distortion

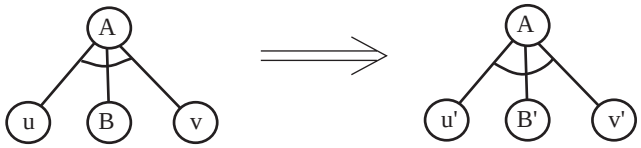
Distortion is an effect caused by the combination of noise and missing data. Distortion is represented as a distinct phenomenon for representation clarity and convenience. The causes and effects of distortion are the same as the causes and effects of noise and missing data. In many domains, distortion is a common phenomenon. For example, in acoustic sensing domains, two low energy peaks that are close to each other in the frequency domain might combine into a single high-energy peak at a frequency between the two original signals.

Definition 4.1.9 *Distortion – A grammar, G, is subject to distortion iff*
$\exists\ p \in P$, *where p:* $A \rightarrow uBv \Rightarrow p' : A \rightarrow n_1 u n_2 B n_3 \mid n_4 B n_5 v n_6 \mid n_7 B n_8 \mid n_9 \lambda n_{10}$, *for u,v* $\in (V \cup N)^*$, $B \in (V \cup N)$, $A \in (SNT \cup N)$, $n_i \in (V \cup N)^*$ *and* $f_p(f_u, f_B, f_v, \Gamma_p(u, B, v)) > f_{p'}$ *for any combination of parameters. In addition, the distribution of* p *and* p' *is modeled by* ψ.



d. The production rule p: A -> uBv and the corresponding `distortion' rule p': A -> $n_1un_2vn_3$.

Figure 4.10. An Example of a Distortion Grammar Rule



e. The production rule p: A -> uBv and the corresponding `masking data' rule p': A -> u'B'v'.

Figure 4.11. An Example of a Masking Grammar Rule

Figure 4.10 is a representation of the definition of distortion. Distortion implies the existence of multiple RHSs for some production rule and, by definition, a distorted rule has a credibility that is lower than the corresponding non-distorted rule. Distortion causes problems that are identical to those caused by both noise and missing data. This includes increasing both the ambiguity of a grammar and the associated cost of problem solving. In addition, *correlated distortion* and *uncorrelated distortion* are phenomena that have definitions identical to correlated and uncorrelated noise and missing data. As with noise and missing data, the correlation of distortion to potentially correct, alternative interpretations defines a continuum in terms of the increased cost required to eliminate incorrect interpretations.

4.1.5 Masking

Masking is a special case of distortion where noise and missing data phenomena combine to generate an RHS that is very similar to the original RHS – for each element e on the RHS of the production rule, p, there is a corresponding element e' on the RHS of p$'$ that has a very similar semantic interpretation. Masking is thought of as a distinct phenomena for intuitive reasons. There are specific, real-world events that lead to masking phenomena in many interpretation domains and these events are significantly different from the events that result in noise, missing data, or distortion. For this reason, masking can be thought of as a unique phenomena rather than a combination of noise and missing data. Formally,

Definition 4.1.10 *Masking – A grammar, G, is subject to masking iff* $\exists$ *a rule* $p \in P$*, where p:* $A \rightarrow uBv \Rightarrow p'$*:* $A \rightarrow u'B'v'$*, for* $u,v,u',v', \in (V \cup N)^*$*,* $A \in (SNT \cup N)$*,* $B,B' \in (V \cup N)$*, and* $f_p(f_u, f_B, f_v, \Gamma_p(u, B, v)) > f_{p'}(f_{u'}, f_{B'}, f_{v'}, \Gamma_{p'}(u', B', v'))$*. In addition, the distribution of* p *and* p' *is modeled by* ψ*.*

Figure 4.11 is a representation of the definition of masking. By definition, a masking data rule has a credibility that is lower than the corresponding non-masking rule. The semantic

interpretation of the masking rule is usually very similar to that of the masked rule. The biggest problem introduced by masking is that it causes correct interpretations to be given lower credibility ratings. This can have detrimental effects when the masked/masking rules are correlated to noise associated with other production rules. In these situations, the noisy rules may have higher credibility values and the problem solver may be forced to expend some amount of work differentiating the alternatives.

4.2 Operator Organization

In Chapter 3, it was specified that the primitive operators available to an interpretation problem solver are represented as rules of the characteristic grammar for the problem solver's domain. In the previous section, inherent uncertainty was defined in terms of additional grammar rules incorporating noise, missing data, masking effects, or some combination of these phenomena. As stated, these additional rules imply the existence of additional problem solving operators.

However, it is possible to organize or group sets of grammar rules into single operators. This is especially useful for efficiently combining large numbers of rules associated with inherent uncertainty and it is done in order to reduce the control costs of invoking operators. By invoking multiple operators as a single unit, it may be possible to reduce overhead costs. In general, two or more grammar rules may be combined into a single *macro-operator*[3] if they are all pairwise syntactically related. Two rules are syntactically related if they have identical left-hand-sides (LHSs). Rules that do not have identical semantics or that are not syntactically related can also be grouped into operators, but such groupings are not considered an operator organization structure.

Operator organization structures can be thought of as the most primitive units of activity that a system reasons about. Operators are usually organized into structures to promote efficiency. Consequently, operator organization can be thought of as a minor transformation to the grammar, often done to offset the effects of inherent uncertainty. Used in this manner, operator organization is very important.

Figure 4.12 shows the notation that will be used to represent operator organizations. Primitive rules will be included in a macro-operator by specifying them as RHS options of a rule (the vertical lines represent "or" notation), or by specifying them as RHS options in "subrules." In Fig. 4.12, the rules labeled 1.b through 1.x are all subrules of 1.a and the RHSs of all these rules are the primitive operators included in macro-operator "1." Essentially, all the primitive rules specified in a macro-operator are "applied" when the macro-operator is invoked.

4.3 Redundancy

Another aspect of interpretation problems that causes difficulty is a special form of ambiguity that is referred to as *redundancy*. Formally,

Definition 4.3.1 *Redundancy – An IDP is subject to redundancy iff, for some input, X, there exist two or more interpretation trees that have identical semantic interpretations for X.*

[3] *When confusion with primitive operators does not result, "macro-operators" will be referred to simply as "operators."*

1. $uAv \rightarrow RHS_1$
2. $uAv \rightarrow RHS_2$
3. $uAv \rightarrow RHS_3$

⋮

m. $uAv \rightarrow RHS_m$

$\Longrightarrow$

1.a. $uAv \rightarrow RHS_1 \mid RHS_2 \mid ... \mid RHS_n$

1.b. $uAv \rightarrow RHS_{n+1} \mid RHS_{n+2} \mid ... \mid RHS_o$

⋮

1.x. $uAv \rightarrow RHS_q \mid RHS_{q+1} \mid ... \mid RHS_m$

Example of an operator organization structure specification.
All RHSs included in 1.a through 1.x are elements of the structure.

Figure 4.12. Example of Operator Organization Representation

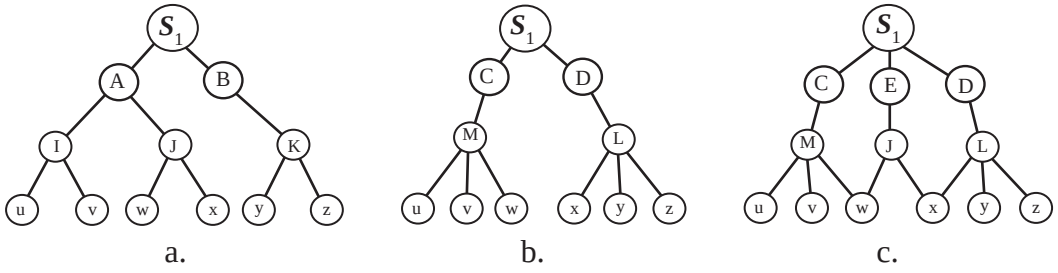


Figure 4.13. Example of Redundancy

The presence of redundancy means that, for at least some interpretations, there are multiple solution paths that lead to the exact same interpretation. For example, consider a system that tracks flying objects using radar and sonic sensors. For any given object, the system can rely on radar or acoustic data (or both) to develop an interpretation. Redundancy provides a problem solver with flexibility in choosing problem-solving activities but also allows results to be rederived using alternative paths, possibly without recognizing the redundancy until the last step. Various studies have shown that the cost associated with redundancy can be substantial [Lesser *et al.*, 1989a]. Figure 4.13 shows an example of redundancy. (Note that this grammar is not the same as the grammar that is being used to illustrate other phenomena. The grammar used in the example of redundancy will not be used in any other examples.)

Redundancy is also shown in Fig. 4.14. This example depicts a stylized version of redundant processing that can occur in certain interpretation tasks such as the Distributed Vehicle Monitoring Testbed (DVMT) [Corkill, 1983]. The problem solving strategy embodied in this example offers alternative paths for interpreting tracks of data, in this case the input "uvwxyz." In Fig. 4.14.b, a track interpretation is formed incrementally by extending a partial interpretation one time unit. Alternatively, in Fig. 4.14.a, a partial interpretation consisting of multiple track positions is extended by combining it with another partial interpretation that also consists of multiple track positions. This type of redundancy is characteristic of convergent search spaces where, without meta-level processing, it is not possible to determine that two search paths lead to the same final interpretation until the paths meet.

1. S → A A
2. S → A T
3. A → A T
4. A → A A
5. A → T T
6. T → u | v | w | x | y | z

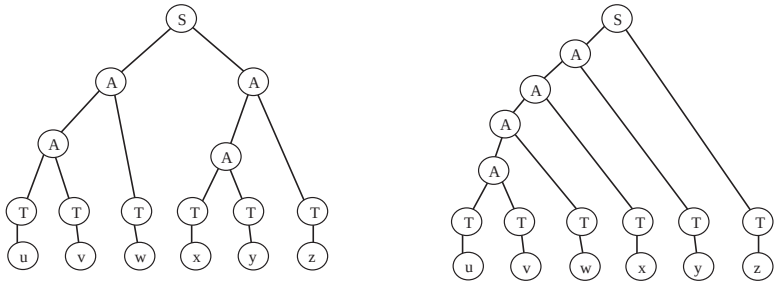


Figure 4.14. Redundant Interpretations for Input "uvwxyz"

4.4 Interacting Subproblems

One of the most important properties that determines a problem's structure is the nature of its interacting subproblems. In the IDP/*UPC* framework, interacting subproblem structures will form the basis for designing, implementing, and analyzing meta-level operations. In particular, relationships between states will be explicitly represented with abstract or approximate states. Recent studies [Lesser *et al.*, 1989a, Lesser *et al.*, 1989b, Decker *et al.*, 1990] have demonstrated that knowledge of a problem's structure can be derived by reasoning about interacting subproblems and that this knowledge can be used to more effectively control problem solving activities. For example, some interacting subproblem structures can be used to dynamically implement hierarchical problem solving strategies, or to implement efficient pruning techniques such as the inhibition of redundant processing [Durfee, 1987, Lesser *et al.*, 1989a, Decker *et al.*, 1990].

In general, control architectures such as the hierarchical problem solving strategies used in [Erman *et al.*, 1980] and [Durfee and Lesser, 1986, Durfee, 1987, Lesser *et al.*, 1989b] can be modeled by transforming an IDP problem representation to include abstractions and approximations used in control actions in the form of meta-operators. The meta-operators will be represented as special rules of the IDP grammar, along with their associated functions, that are derived from interacting subproblems in the base IDP model.

In domains with interacting subproblem structures, control architectures that are based on "local control" (from Chapter 2.2) strategies are often ineffective when compared to control architectures that are capable of exploiting the subproblem interactions. However, this is somewhat misleading. Unlike uncertainty and redundancy, interacting subproblems do not *increase* the complexity or difficulty of interpretation problems. Rather, their importance is derived from their ability to *reduce* the cost of problem solving dramatically. Consequently, the need to formally specify structures associated with interacting subproblems is motivated by the desire to exploit these structures and not by the need to avoid or compensate for their presence.

To understand these observations, consider two very similar grammars G_1, with interacting subproblems, and G_2, with no interacting subproblems. Assume that the complexity of the domains described by the grammars is equivalent from the perspective of a common local control architecture. In other words, the local control architecture does not attempt to exploit the interacting subproblem structures in G_1. In this situation, the interacting subproblems in G_1 do not increase the complexity of problem solving in the corresponding domain. However, a control architecture can be constructed to exploit the interacting subproblems and a problem solver using such a control architecture would likely outperform a problem solver using only a local control architecture. This would depend on the costs associated with recognizing interacting subproblems, the accuracy with which the effects of interrelationships are evaluated, the strength with which these interrelationships constrain further problem solving, and the cost of exploiting these relationships.

Though the majority of discussion related to interacting subproblems is contained in later chapters and other publications [Lesser *et al.*, 1989b], their definitions and representations are included here for consistency.

4.4.1 Defining Interacting Subproblems

Formal definitions of interacting subproblems will be based on the following definitions of *component set* and *result set*. The definitions given here will rely on interpretation grammar G' from Fig. 3.2, on extensions to G' presented in Fig.4.4, and on Figures 4.15, 4.16, and 4.17. In these figures, the states labeled A' and D' are indications of multiple instantiations of the states A and D. They do not represent new or unique states.

Definition 4.4.1 *Component Set (CS) - The component set of a state, s_n, includes all the states that lie on paths from the signal data to s_n. Figure 4.15 shows the complete convergent search space associated with an interpretation of A as well as paths to a state corresponding to an interpretation of M. Figure 4.16 shows the component set of an intermediate state labeled C. An intermediate search state such as C corresponds to a partial interpretation tree, or* interpretation subtree, *t. The component set of a state includes its corresponding interpretation tree or subtree, t.*

Definition 4.4.2 *Result Set (RS) - The result set of a state, s_n, in a search space includes all the states that can be reached from s_n. Figure 4.17 shows the result set of the intermediate state q.*

Definition 4.4.3 *Subproblem - In the IDP framework, a subproblem is any symbol from the alphabet of of an IDP grammar's that appears on the right-hand-side of a production rule.*

Definition 4.4.4 *Subproblem Relationships and Interactions - If a function, r, is defined in terms of the Component Sets and/or Result Sets of two or more subproblems, we say that r defines a* relationship *(or* interaction*) between the subproblems.*

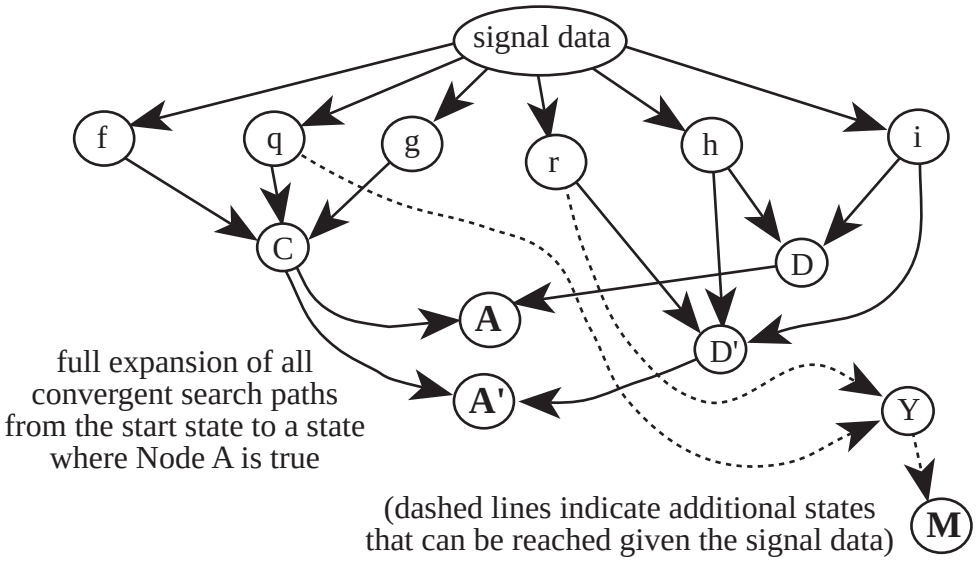


Figure 4.15. Example of a Fully Expanded Convergent Search Space

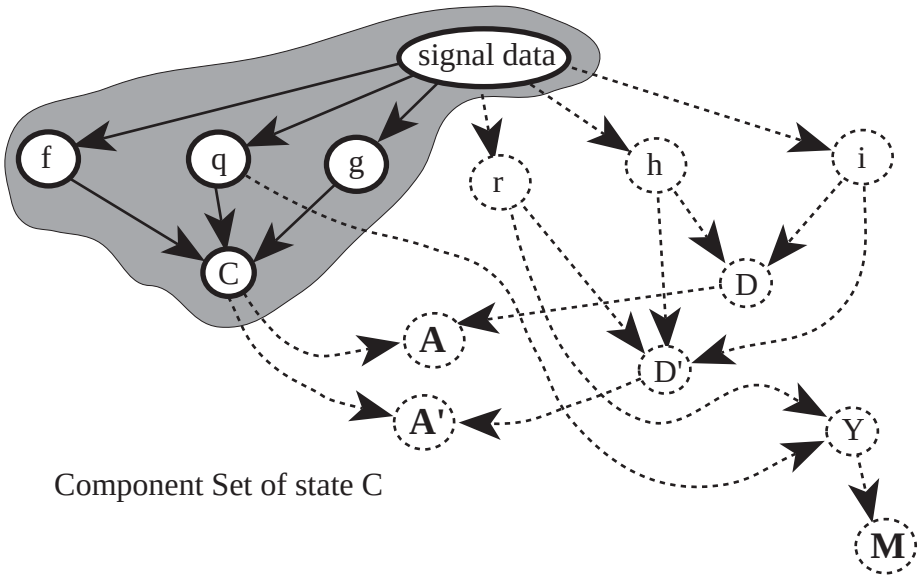


Figure 4.16. Component Set Example

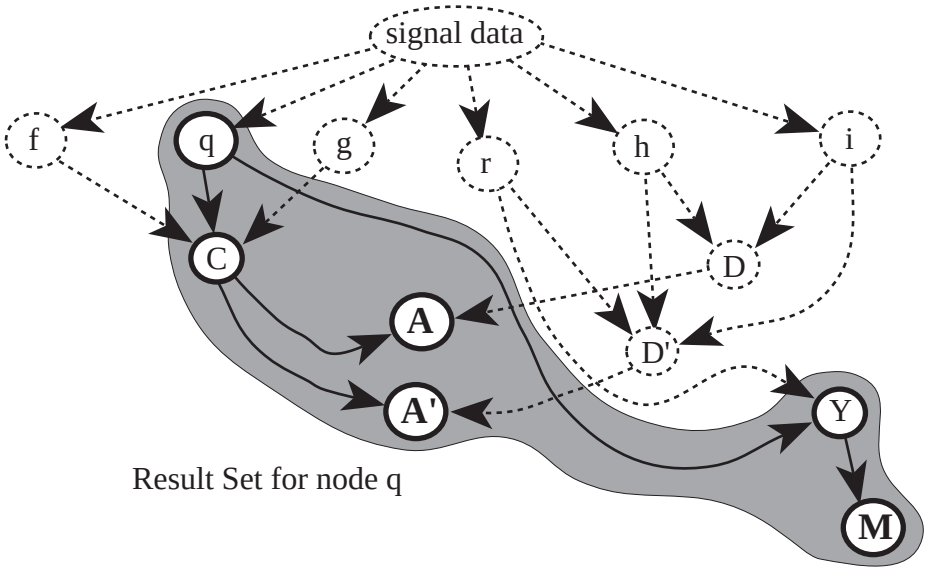


Figure 4.17. Result Set Example

In [Lesser *et al.*, 1989a] a number of relationships were defined and their use in controlling problem solving was discussed. These definitions are now updated.[4]

Definition 4.4.5 *Subsumption: Partial result* t_1 *subsumes a second partial result,* t_2*, if the component set of* t_2 *is a subset of the component set of* t_1*. Formally,* t_1 *subsumes* t_2 *if* $\forall y, y \in CS(t_2) \Rightarrow y \in CS(t_1)$.

Definition 4.4.6 *Competition: Two partial results,* t_1 *and* t_2*, are competing if the intersection of their component sets is non-empty and the intersection of their result sets is empty. Partial results* t_1 *and* t_2 *are competing iff* $CS(t_1) \cap CS(t_2) \neq \emptyset$ *and* $RS(t_1) \cap RS(t_2) \neq \emptyset$.

Definition 4.4.7 *Cooperation: Two partial results are cooperating if the intersection of their component sets is empty and the intersection of their result sets is non-empty. In other words, they are cooperating if it is possible that they might be incorporated into a single interpretation tree at some point in the future. Partial results* t_1 *and* t_2 *are* completely cooperating *iff* $RS(t_1) = RS(t_2)$ *(i.e.,* $RS(t_1) \cap RS(t_2) = RS(t_1) \cup RS(t_2)$*. Partial results* t_1 *and* t_2 *are* partially cooperating *iff* $RS(t_1) \cap RS(t_2) \neq \emptyset$ *and* $(RS(t_1) \cap RS(t_2)) \neq (RS(t_1) \cup RS(t_2))$.

Definition 4.4.8 *Overlapping: Two partial results are overlapping if the intersection of their component sets is non-empty and the intersection of their result sets is non-empty. In addition, their component sets must not be identical. In other words, they are overlapping if they share some component subtrees and if it is possible that they might be incorporated into a single interpretation tree at some point in the future. Partial results* t_1 *and* t_2 *are overlapping iff* $CS(t_1) \neq CS(t_2)$, $CS(t_1) \cap CS(t_2) \neq \emptyset$ *and* $RS(t_1) \cap RS(t_2) \neq \emptyset$.

[4] The term "partial result" will refer to a partial interpretation tree.

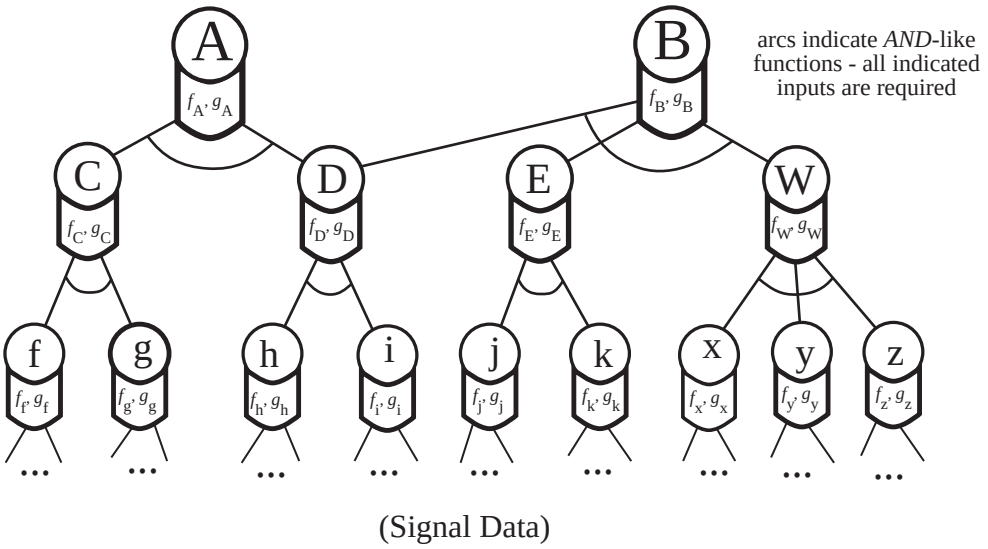


Figure 4.18. Graphical Representation of Example Interpretation Grammar

Definition 4.4.9 *Independence: Two partial results are independent if the intersection of their component sets is empty and the intersection of their result sets is empty. This means that they are not competing and that it is not possible for them to be incorporated into a single interpretation tree other than an interpretation rooted at the signal data. Partial results* t_1 *and* t_2 *are independent iff* $CS(t_1) \cap CS(t_2) = \emptyset \ \wedge RS(t_1) \cap RS(t_2) = \emptyset$.

4.4.2 Representing Interacting Subproblems With the IDP Model

In the IDP formalism, interacting subproblem structures will be explicitly represented by abstract states. The relationships captured by these abstract states will be defined in terms of the relationships from the previous section. These relationships will be specified by the addition of grammar rules that will be referred to as *meta-operators*, *meta-rules*, or *abstract operators*. Thus, meta-operators and meta-rules are problem solving actions that either create or modify abstract states. In addition, abstract states will be associated with mapping functions that extend or modify the base space in some way in order to use the results of meta-level processing. The following example illustrates how the IDP formalism can represent interacting subproblems and the associated meta-operators and mapping functions. The IDP representation of meta-level processing is used to implement a variety of problem solving strategies, especially top-down processing (branch and bound) strategies.

Figure 4.19 shows an example of meta-operators for the grammar G' from Fig. 4.18. This figure is repeated because it naturally represents certain subproblem relationships. This example is intended to illustrate the representation techniques that are used in the IDP framework and it is not intended to demonstrate the utility of the techniques.

In this example, rules M5 and M6 are added indicating that the abstract, "meta-state" M can be derived from an h or an i. Rules M3 and M4 indicate that the abstract states A' and B' can be derived, respectively, from an f (with M) or an x (also with M) respectively. Rules M1 and M2 indicate, respectively, that an A can be derived from an A' and a B can be derived from

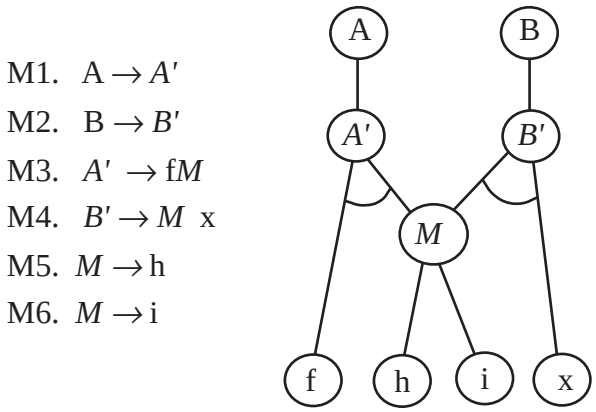


Figure 4.19. Meta-Operators Expressed as Rules of a Grammar

a B'. Note that these rules are additions to the interpretation grammar, IDP_i, and they are not used for generational purposes, only for representing the structure of the problem solver. They are not used to model the events that created the observed signal data and these rules would be ignored by a domain event simulator that is based on the use of an IDP grammar specification.

From the grammar in Fig. 4.18, it should be clear that h and i are completely cooperating. Consequently, there is no point in pursuing distinct solution paths from h and i - there is no interpretation that can be formed starting with i that will not be formed starting with h, and vice-versa. The abstract state M represents this relationship. Intuitively, M can be thought of as a combination of the paths from h and i.

It should also be clear from the grammar that f and x are independent and that f and x are both cooperating with M. The abstract states A' and B' represent these relationships. A' represents the intersection of the Result Sets of f and M and B' represents the intersection of the Result Sets of x and M.

In order to exploit the relationships represented by the abstract states A', B' and M, there must be a corresponding mapping function that, when applied to one of the abstract states, extends or modifies the base space. This role is filled by the mapping functions corresponding to rules M1 and M2.

Rules M5 and M6 can be thought of as meta-operators that project the base space to an abstraction space by generating an abstract state, M, given an h or an i. Rules M3 and M4 can be thought of as top-down operators that bound M. For example, M3 checks for the presence of an f, in which case it generates an A'. If there is no f, M3 fails and generates no new states. Similarly, M4 checks for the presence of an x. If there is such data, M4 will generate a B', otherwise it will fail and not generate any new states.

The rules M1 and M2 map the projection space back to the base space. Given an A', the problem solver knows that the correct interpretation is an A, and it adjusts the *UPC* values of low-level states appropriately. Unlike the other grammar rules, mapping operators do not create new states in the base space[5]. The production rule notation is used to specify the states

[5] This convention is used for the discussion and analysis presented in this thesis.

that are updated, and the information used to update them. Specifically, the states that are updated are those in the component set of the abstract state. These states are updated with information about the final state that is on the left-hand-side of the grammar rule specifying the mapping operator. For example, meta-operator M1 will update the *UPC* values for states M, f, h, and i with information indicating that the probability of generating an A is 1. It is important to note that, even though the problem solver knows the interpretation has to be an A, it still has to execute all the semantic functions to develop the correct "meaning" of the signal data. For example, in a vehicle tracking domain, the problem solver may know that the vehicle it is tracking is of type 1, but it will still have to execute the semantic functions to develop a full interpretation of the vehicle's actions over a period of time. Similarly, given a B', the meta-operator M2 updates the *UPC* values for states M, h, i, and x with information indicating that the probability of generating a B is 1.

The advantages of adding the meta-operators shown in Fig. 4.19 can be analyzed with the tools that are defined later in this thesis. The data needed for proper analysis of the value of these meta-operators has been omitted, but, intuitively, the meta-rules have obvious advantages in terms of reducing the superficial complexity of the problem. Use of these meta-operators will eliminate the application of many redundant processing steps, such as extending search paths from the lower-level states that will eventually turn out to be redundant, and it is reasonable to expect that this will reduce the overall cost of problem solving. Though this example was kept intentionally simple to illustrate the representation used in the IDP formalism, subsequent examples in will be based on very similar analysis.

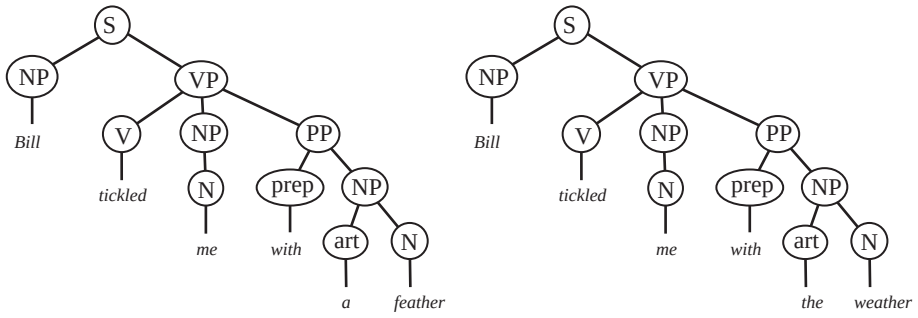
4.5 Non-Monotonicity and Bounding Functions

Monotonicity is a property of problem structures that has been used to construct many important search control architectures [Kumar and Kanal, 1988]. Though there is no guarantee that a monotone problem can be solved efficiently, most existing efficient control architectures such as AO*, alpha-beta, B*, SSS* and their generalizations are only applicable in monotone domains. In addition, problem solving techniques such as dynamic programming, A* search, Martelli and Montanari's bottom-up search, and Dijkstra's shortest path algorithm are also restricted to monotonic domains.

For the most part, IDP structures of interest will be non-monotone. However, even in non-monotone structures, it is still possible to define *bounding functions* that can be used effectively in control architectures. The structures associated with these functions are defined in this chapter.

4.5.1 Non-Monotonicity

An interpretation problem's monotonicity is specified in terms of the function f_p associated with production rule $p : n \rightarrow n_1 \ldots n_k$. If, $\forall p$, when all of the k-ary credibility attributes associated with the production $p : n \rightarrow n_1 \ldots n_k$ are monotonically nondecreasing in each variable it is implied that the corresponding semantic functions and evaluation functions are also nondecreasing, then the interpretation problem is said to be monotonic. i.e., if $x_1 \leq y_1 \& \ldots \& x_k \leq y_k$ implies $(\Gamma_p(x_1 \ldots x_k) \leq \Gamma_p(y_1 \ldots y_k)$ and $f_p(f_{x_1} \ldots f_{x_k}, \Gamma_p(x_1 \ldots x_k)) \leq f_p(f_{y_1} \ldots f_{y_k}, \Gamma_p(y_1 \ldots y_k)))$, where x_i and y_i represent the i^{th} element of the rule p, and function Γ_p is the semantic credibility function associated with p, then the interpretation problem is monotone.



$$f_S(\text{“Bill tickled me with a feather.”}) > f_S(\text{“Bill tickled me with the weather.”})$$

$$f_{VP}(\text{“tickled me with a feather”}) < f_{VP}(\text{“tickled me with the weather”})$$

$$f_{PP}(\text{“with a feather”}) < f_{PP}(\text{“with the weather”})$$

Figure 4.20. An Example of a Non-Monotone Interpretation Domain

For many interpretation domains, such as the DVMT[Corkill, 1983] and Hearsay-II[Erman *et al.*, 1980], monotonicity does not hold. A simple example illustrates how this might occur. Figure 4.20 shows two interpretations of signal data in a speech understanding domain. In this example, the partial interpretation “with the weather” is rated much higher than the partial interpretation “with a feather” as the result of processing the low-level acoustic information. The rating function does not have a broader context in which to interpret the signal data, so it relies more heavily on the purely acoustic properties. The strength of the acoustic based rating is strong enough that “tickled me with the weather” is rated higher than the partial interpretation “tickled me with a feather.” This is typical of a monotonic domain – the same data was used to extend the original partial interpretations and the processing maintained the rank order of the resulting credibilities. i.e., the low-level result with the highest rating still has the highest rating after all low-level results are extended with *the same data.* In a monotone domain, this would also mean that the full interpretation “Bill tickled me with the weather.” would be the highest rated solution. This is because $f_{NP}(\text{Bill}) = f_{NP}(\text{Bill})$, which, by the preceding definition of a monotone IDP, would imply that $f_S(\text{“Bill tickled me with a feather.”}) \leq f_S(\text{“Bill tickled me with the weather.”})$. However, in this example, the best full interpretation of the signal data is “Bill tickled me with a feather.” When forming this interpretation, the problem solver has a broader context in which to evaluate the semantic consistency of the input and the importance of the low-level acoustic properties is diminished. Consequently, this is not a monotone domain.

The non-monotonicity of a domain can be thought of as a function of the semantics of a domain. As individual components are added to a growing interpretation, they must be consistent with all of the other components that have been added so far. In many interpretation domains, such as speech recognition, components with high individual credibilities are often inconsistent with some or all of the other components of a partial interpretation [Erman *et al.*, 1980]. For example, a particular word may be the best interpretation for a particular time-slice of data, but that word may not be consistent with the “meaning” (i.e., the semantics) of a partial interpretation the problem solver is trying to extend. As a consequence, the resulting

extension may have an arbitrarily low-rated credibility depending on how ridiculous it is.

4.5.2 Bounding Functions

In certain monotone domains, problem solving is simplified by the fact that optimal subproblem solutions are guaranteed to be components of the optimal solution [Kumar and Kanal, 1988]. In non-monotone interpretation problems, this guarantee obviously does not hold. However, based on the problem structure defined by a domain's characteristic grammar, G, and evaluation function f, certain *bounding functions* can be defined. For example, given a partial interpretation, i, a bounding function determines the likelihood of i being included in a solution with a rating above some *threshold*. If it is unlikely that the partial interpretation will lead to a solution rated above the threshold, all derivation paths including the partial interpretation are pruned. For example, given threshold t, for any partial interpretation, i, if $upperbound(i) < t$, then i can be pruned. $upperbound(i)$ can be defined as the maximum expected utility for any of the final states that can be reached along paths from i. Bounding functions are also referred to as *pruning functions* or *pruning operators*. *Static pruning functions* are those that use a predefined threshold that does not vary during problem solving. *Dynamic pruning functions* use a threshold that is determined during problem solving. In the experiments in Chapter 11, a dynamic pruning threshold corresponds to the rating of a full interpretation.

Intuitively, the threshold can be established in a number of ways. One way is to set it at a level where any solution rated below the threshold would be considered very questionable and the problem solver would not return any such ratings. Another way to set the threshold is to examine the distribution characteristics of interpretations for a domain and set the threshold at a level where there is a very high probability that at least one solution will have a rating above the threshold. In an ideal situation, only a single solution will have a rating above the threshold. For example, in the Hearsay II speech understanding system, experiments were conducted using a "tight" grammar and a vocabulary restricted to 250 words. The termination criterion given these parameters was to halt processing when the first solution above a threshold was generated. However, in a situation where the system parameters included a "loose" grammar and a 1,000 word vocabulary, the results generated by this termination criterion were unacceptable. This set of experiments demonstrated that tight grammars/small vocabularies lead to less ambiguity than loose grammars/large vocabularies [Erman *et al.*, 1980].

Like interacting subproblems, bounding functions and monotone properties are structures that can be exploited by a control architecture to reduce the overall cost of problem solving. And similar to interacting subproblems, the importance of identifying these structures is based on this potential to increase the efficiency of problem solving.

In addition to structures specified by bounding functions, non-monotone domains often exhibit *semi-monotone structures*. These are structures where the rating of a partial interpretation is correlated to the probability that the partial interpretation is a component of the best interpretation. Thus, partial interpretations with relatively low ratings have a low probability of being a component of correct solution and partial interpretations with relatively high ratings have a high probability of being a component of correct solutions. In semi-monotone domains, it is possible to prune paths based on the credibility of intermediate results in such a way that the probability of eliminating the correct interpretation is known. This strategy can be used in situations where it is possible to gain dramatic performance improvements by accepting a slight risk of eliminating the correct solution from consideration. This is discussed in more depth in Chapters 7 and 11.

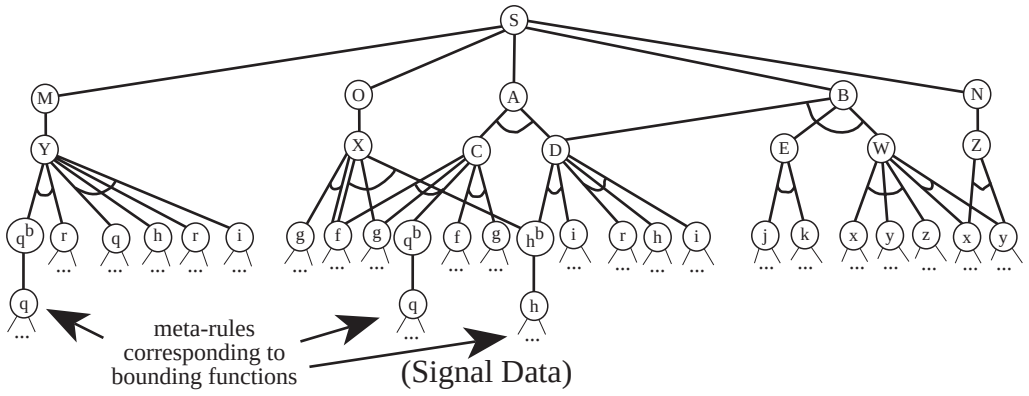


Figure 4.21. Example of Bounding Function Incorporated in a Grammar

4.5.3 Representing Bounding Functions With the IDP Model

In the IDP formalism, bounding functions are represented by extending the interpretation grammar, IDP$_i$. Given a state, n, for which a bounding function can be defined, every appearance of n on the RHS of a grammar rule in IDP$_i$ is replaced with the nonterminal n^b and the rule $p : n^b \rightarrow n$ is added to the grammar. The knowledge incorporated in the operator corresponding to p is the bounding function on n. Given an n, the operator p executes the bounding function. If the output of the bounding function indicates that extending interpretations using state n is pointless, the operator fails to generate n^b and no further derivations using n are built. If the output of the bounding function is within a range indicating that further interpretations using n should be constructed, the operator will generate n^b and processing will continue. More specifically, in the IDP/*UPC* framework, a state is pruned by the credibility function, f_{n^b}, which returns a value of 0 for that state. As will be discussed in a later section which provide details of the IDP/*UPC* testbed, states with credibility of 0 are not expanded.

In this representation, the context the bounding function operates in, the "bounding context," is not represented explicitly in the grammar. Instead, the bounding context is specified in the definition of the evaluation function, f_{n^b}, associated with bounding rule p. This is due to a lack of a satisfactory representation scheme that would enable the bounding context to be represented explicitly in the grammar in a clear and unambiguous manner.

An example of incorporating bounding functions in a grammar is shown in Fig. 4.21. In this example, a bounding function has been defined for state q, and the implications have been incorporated in the extended version of grammar G' shown in Fig. 4.3. The inclusion of a bounding function is represented by the addition of the rule $q^b \rightarrow q$. If, when applied, the bounding function operator determines that there is no point in extending state q, it will fail to generate state q^b. This will prevent the generation of state Y and make the application of the operator corresponding to the rule $C \rightarrow fgq^b$ unnecessary. This bounding function operator might be useful in a situation where state q is generated as the result of noise and has a credibility rating so low that any interpretation incorporating it would have an extremely low rating.

Bounding functions incur a cost to execute and, as a consequence, there is a corresponding increase in the expected cost of problem solving when a grammar is extended to include

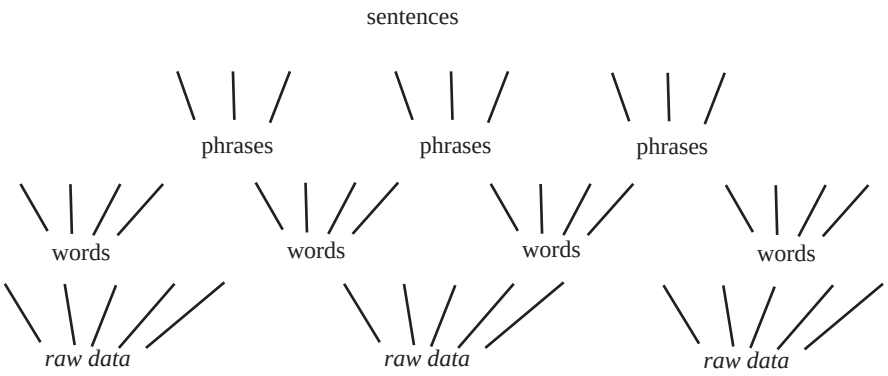


Figure 4.22. Example of a Natural Language Processing System

bounding functions. This increased cost is offset by situations where a bounding function eliminates paths from further consideration and the corresponding costs are not incurred.

To better understand the role of bounding functions, consider the case of a natural language understanding system where low-level data is processed into word elements, word elements are processed into phrase elements, and phrase elements are processed into sentences. A representation of this system is shown in Fig. 4.22. Furthermore, assume that analysis has indicated that words with credibilities less than 0.33 (on a scale of 0 to 1.0) have probability = 0 of being included in a correct interpretation.

In this situation, it may be advantageous to add bounding functions to the grammar to check to see if a word element has a credibility greater than 0.33 before trying to extend the corresponding search state. If a state corresponding to a word has a credibility less than 0.33, then any effort expended trying to extend this state is wasted effort and pruning the state without expanding might reduce the overall cost of problem solving. Whether or not this is advantageous is based on a number of factors, but the general rule is that if the overall cost of executing the bounding functions is less than the cost associated with the pruned search paths, then including bounding functions is advantageous.

For example, consider a case where the expected cost of expanding a state corresponding to a word with credibility less than 0.33 is 10, where the probability of a word having a credibility less than 0.33 is .75, and where the expected quantity of words generated in a problem instance is 100. In this case, the expected cost associated with searching paths from "bad" states, i.e., states corresponding to words that have probability = 0 of being included in a correct interpretation, is $0.75 * 10 * 100 = 750$.

Now, assume that bounding functions are added that prune states with credibilities less than 0.33. If the expected cost of each bounding function is 10, the overall cost associated with the bounding functions will be cost multiplied by the expected number of words that the bounding functions are applied to, or $10 * 100 = 1000$. In other words, the cost of applying the bounding functions would be 1000, and, from the previous equation, the cost without the use of the bounding functions would be 750. Thus, the use of bounding functions would increase the expected cost of problem solving by 250 units.

Now consider if the expected cost of each bounding function is 5. In this case, the cost of applying the bounding functions is $5 * 100 = 500$. Thus the use of bounding functions reduces the expected cost by 250 units.

To illustrate another issue that must be considered in the analysis of bounding functions, consider the situation where the expected cost of applying each bounding function is 5, the expected number of words generated in each problem solving instance is 100, the expected cost of searching a bad path is 10, all as in the previous situation, but the probability of a word having credibility less than 0.33 is 0.25. In this case, the expected cost of applying the bounding functions will still be the expected cost of each bounding function application multiplied by the expected number of words, $5 * 100 = 500$, but the expected cost savings from pruning bad paths will be the probability that a word has a credibility less than 0.25 multiplied by the number of words multiplied by the expected cost of a bad path, or $0.25 * 100 * 10 = 250$. In this situation, the net effect of using bounding functions is to increase the expected cost of problem solving by 250 units.

In the IDP/*UPC* framework, explicitly representing bounding functions as an integral part of the set of problem solving actions facilitates analysis such as that presented in the preceding paragraphs. In addition, the IDP/*UPC* framework can be used to analyze more sophisticated bounding function implementations where the thresholds are determined dynamically. This will be demonstrated further in Chapter 8.

4.6 Generating Problem Instances with the Feature List Convention

To demonstrate how an IDP grammar can be used to represent a complex, real-world domain, we will first describe how an IDP grammar is used to generate problem instances. For this purpose a complex grammar that is used in Chapter 11 is introduced here. It will be used in this and the following sections to illustrate important aspects of the IDP/*UPC* framework. The interpretation domain that will be studied in detail in Chapter 11 is a vehicle tracking problem as defined in previous work on distributed vehicle monitoring in the RESUN research project [III, 1990] and in the Distributed Vehicle Monitoring Testbed (DVMT) [Corkill, 1983]. The problem solver's input consists of preprocessed sensor data gathered from a single contiguous region. The problem solver then processes the data in an attempt to identify the type of vehicle that generated the signals and the path it traversed through the region.

The preprocessed signal data used in the experiments is generated by a grammar based mechanism. Figure 4.23 represents the basic approach used to generate problem instances. As shown in the figure, each symbol of the grammar has a feature list. In addition, each production rule is associated with a set of functions which take a feature list as an input.

In order to generate a problem instance, the simulator begins by creating a feature list for the start symbol, S. The most important aspect of this task is the determination of a credibility. Each grammar has an expected credibility and variance, and this information is used to probabilistically choose a specific credibility for the problem instance. In addition, other features can be determined at this time. For example, in a vehicle tracking domain, some of the features may include vehicle heading, velocity, type, etc.

Next, the generator probabilistically chooses a production rule, p, associated with the start symbol and applies p to the start symbol. i.e., the generator randomly chooses a production rule with the start symbol on the left-hand-side and replaces S with the right-hand-side of the rule. The choice of which rule to apply is made based on the weights of the ψ associated with the different rules. As the generator replaces S, it assigns each of the replacing symbols a feature list. Each of the feature lists is determined by applying p's feature list functions to S's original feature list.

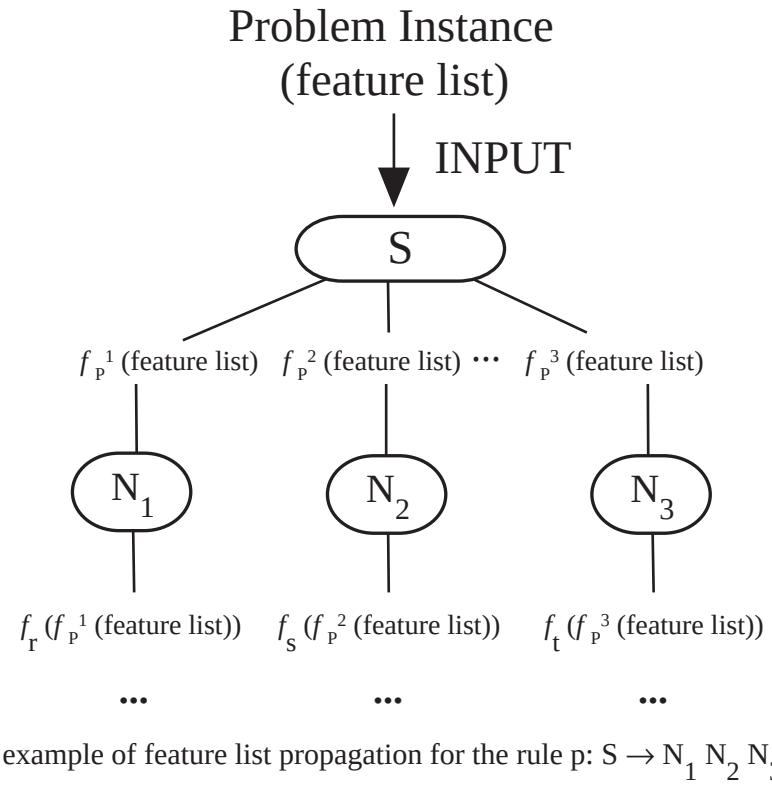


Figure 4.23. Generating Problem Instances with the Feature List Convention

The new symbols, each with their own, unique feature list, are added to either a queue, if the symbol is a nonterminal, or an output set, if the symbol is a terminal, and the whole process is repeated. The first element of the queue is removed and processed, and the resulting new symbols are added to the tail of the queue or to the output set. This continues until the queue is empty and the output set contains terminals with fully specified feature lists.

The basic structure of the problem domain is shown in Fig. 4.24. As shown, the domain consists of *scenarios*. Each scenario consists of a number of *tracks*. Each track can represent a single vehicle traversing the region (an *I-Track*), a vehicle and a corresponding ghost track (a *G-Track*), or a group of vehicles moving in a coordinated manner (a *P-Track*). Each of these tracks is composed of *vehicle locations*. Vehicle locations are composed of *group classes* and group classes consist of *signal classes*. The signal classes can be thought of as preprocessed signal data [Corkill, 1983].

A grammar that will generate problem instances for this domain is shown in Figures 4.25, 4.27, 4.29, and 4.30. These figures show the production rules of the grammar and the probability distribution values for ψ associated with each of the rules. The feature lists are shown as bracketed subscripts. The key to interpreting the feature lists and functions of feature lists is the following [Whitehair and Lesser, 1993]:

f: A feature list. In this example, "f" represents characteristics that are not otherwise represented such as energy, frequency, etc. [6]

[6] In the examples presented in this thesis, "f" is included to indicate that additional characteristics can be added

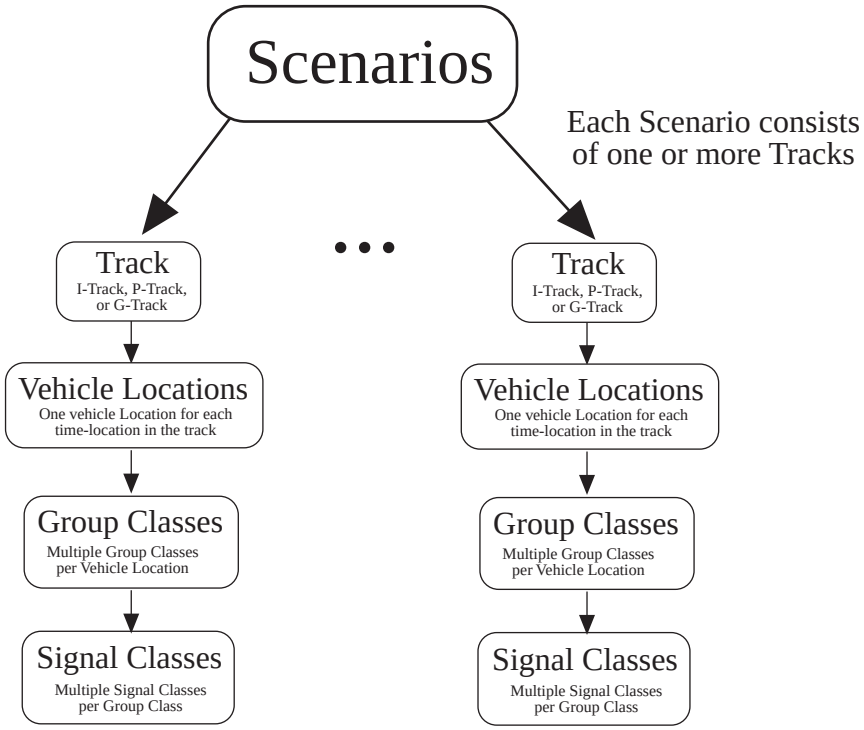


Figure 4.24. The Basic Structure of the Vehicle Tracking Problem Domain

t: Time.

x: The x-coordinate of a location.

y: The y-coordinate of a location.

vel: Velocity. This is used to generate the x and y coordinates of the next location in the track.

acc: Acceleration. This is used to adjust velocity over time. Velocity and acceleration are also used to constrain the progression of a track. For example, each particular kind of vehicle track has a maximum velocity and acceleration that cannot be exceeded.

offset: The offset from a "true" position. This is used to generate pattern tracks of multiple vehicles and ghost tracks.

A problem instance is created starting with the generation of the start symbol, S, and its feature list. The most important aspect of this task is the determination of a credibility. Each grammar has an expected credibility and variance, and this information is used to probabilistically choose a specific credibility for the problem instance. Initially, the feature list consists solely of an "energy" level, which is represented as part of "f" in the grammar shown.

---

if desired. Our experimental system is implemented so that additional features can be added with little effort. In some cases, we have modified "f" to represent other vehicle tracking domain characteristics in order to test the adequacy of the representational power of the IDP formalism. For the sake of clarity, we do not treat their representation in more detail.

1.	$S_{[f]}$	$\rightarrow$ $Tracks_{[f]}$	p=1
2.	$Tracks_{[f]}$	$\rightarrow$ $Tracks_{[f]}$ $Track_{[f]}$	p=0
		$\rightarrow$ $Track_{[f]}$	p=1
3.	$Track_{[f]}$	$\rightarrow$ $I\text{-}Track1_{[f,t,x,y]}$	p=0.25
		$\rightarrow$ $I\text{-}Track2_{[f,t,x,y]}$	p=0.25
		$\rightarrow$ $P\text{-}Track1_{[f,t,x,y]}$	p=0.10
		$\rightarrow$ $P\text{-}Track2_{[f,t,x,y]}$	p=0.10
		$\rightarrow$ $G\text{-}Track1_{[f,t,x,y]}$	p=0.15
		$\rightarrow$ $G\text{-}Track2_{[f,t,x,y]}$	p=0.15
4.	$I\text{-}Track1_{[f,t,x,y]}$	$\rightarrow$ $I\text{-}Track1_{[f,t+1,x+vel+acc,y+vel+acc]}$ $T1_{[f,t,x,y]}$	p=1
5.	$I\text{-}Track2_{[f,t,x,y]}$	$\rightarrow$ $I\text{-}Track2_{[f,t+1,x+vel+acc,y+vel+acc]}$ $T2_{[f,t,x,y]}$	p=1
6.	$P\text{-}Track1_{[f,t,x,y]}$	$\rightarrow$ $P\text{-}Track1_{[f,t+1,x+vel+acc,y+vel+acc]}$ $P\text{-}T1_{[f,t,x,y]}$	p=1
7.	$P\text{-}Track2_{[f,t,x,y]}$	$\rightarrow$ $P\text{-}Track2_{[f,t+1,x+vel+acc,y+vel+acc]}$ $P\text{-}T2_{[f,t,x,y]}$	p=1
8.	$G\text{-}Track1_{[f,t,x,y]}$	$\rightarrow$ $G\text{-}Track1_{[f,t+1,x+vel+acc,y+vel+acc]}$ $G\text{-}T1_{[f,t,x,y]}$	p=1
9.	$G\text{-}Track2_{[f,t,x,y]}$	$\rightarrow$ $G\text{-}Track2_{[f,t+1,x+vel+acc,y+vel+acc]}$ $G\text{-}T2_{[f,t,x,y]}$	p=1
10.	$P\text{-}T1_{[f,t,x,y]}$	$\rightarrow$ $T1_{[f,t,x+offset,y+offset]}$ $T2_{[f,t,x,y]}$	p=1
11.	$P\text{-}T2_{[f,t,x,y]}$	$\rightarrow$ $T2_{[f,t,x+offset,y+offset]}$ $T2_{[f,t,x,y]}$	p=1
12.	$G\text{-}T1_{[f,t,x,y]}$	$\rightarrow$ $GT1_{[f,t,x+offset,y+offset]}$ $T1_{[f,t,x,y]}$	p=1
13.	$G\text{-}T2_{[f,t,x,y]}$	$\rightarrow$ $GT2_{[f,t,x+offset,y+offset]}$ $T2_{[f,t,x,y]}$	p=1
14.	$T1_{[f,t,x,y]}$	$\rightarrow$ $V1_{[f,t,x,y]}$ $N_{[f,t]}$	p=1
15.	$T2_{[f,t,x,y]}$	$\rightarrow$ $V2_{[f,t,x,y]}$ $N_{[f,t]}$	p=1
16.	$GT1_{[f,t,x,y]}$	$\rightarrow$ $GV1_{[f,t,x,y]}$ $N_{[f,t]}$	p=1
17.	$GT2_{[f,t,x,y]}$	$\rightarrow$ $GV2_{[f,t,x,y]}$ $N_{[f,t]}$	p=1
18.	$N_{[f,t]}$	$\rightarrow$ $n_{[f,t]}$ $N_{[f,t]}$	p=0.1
		$\rightarrow$ $n_{[f,t]}$	p=0.25
		$\rightarrow$ λ	p=0.65
19.	$V1_{[f,t,x,y]}$	$\rightarrow$ $G1_{[f,t,x,y]}$ $G3_{[f,t,x,y]}$ $G7_{[f,t,x,y]}$	p=0.4
		$\rightarrow$ $G1_{[f,t,x,y]}$ $G3_{[f,t,x,y]}$	p=0.3
		$\rightarrow$ $G1_{[f,t,x,y]}$ $G7_{[f,t,x,y]}$	p=0.25
		$\rightarrow$ λ	p=0.05
20.	$V2_{[f,t,x,y]}$	$\rightarrow$ $G3_{[f,t,x,y]}$ $G8_{[f,t,x,y]}$ $G12_{[f,t,x,y]}$	p=0.4
		$\rightarrow$ $G8_{[f,t,x,y]}$ $G12_{[f,t,x,y]}$	p=0.3
		$\rightarrow$ $G3_{[f,t,x,y]}$ $G12_{[f,t,x,y]}$	p=0.25
		$\rightarrow$ λ	p=0.05
21.	$GV1_{[f,t,x,y]}$	$\rightarrow$ $G\text{-}G1_{[f,t,x,y]}$ $G\text{-}G3_{[f,t,x,y]}$ $G\text{-}G7_{[f,t,x,y]}$	p=0.2
		$\rightarrow$ $G\text{-}G1_{[f,t,x,y]}$ $G\text{-}G3_{[f,t,x,y]}$	p=0.3
		$\rightarrow$ $G\text{-}G1_{[f,t,x,y]}$ $G\text{-}G7_{[f,t,x,y]}$	p=0.25
		$\rightarrow$ λ	p=0.05
22.	$GV2_{[f,t,x,y]}$	$\rightarrow$ $G\text{-}G3_{[f,t,x,y]}$ $G\text{-}G8_{[f,t,x,y]}$ $G\text{-}G12_{[f,t,x,y]}$	p=0.4
		$\rightarrow$ $G\text{-}G8_{[f,t,x,y]}$ $G\text{-}G12_{[f,t,x,y]}$	p=0.3
		$\rightarrow$ $G\text{-}G3_{[f,t,x,y]}$ $G\text{-}G12_{[f,t,x,y]}$	p=0.25
		$\rightarrow$ λ	p=0.05

Figure 4.25. Grammar Rules for Generating Patterns and Tracks

Next, the generator probabilistically chooses a production rule, p, associated with the start symbol and applies p to the start symbol. i.e., the generator randomly chooses a production rule with the start symbol on the left-hand-side and replaces S with the right-hand-side of the rule. The choice of which rule to apply is made based on the weights of the ψ associated with the different rules. As the generator replaces S, it assigns each of the replacing symbols a feature list. Each of the feature lists is determined by applying p's feature list functions to S's original feature list.

Specifically, from the start symbol, grammar transformations using the rules shown lead to the application of one of the rules 4 through 9. At this point, the next selection of a right-hand-side (RHS) of the grammar to be used determines the *type* of track that will be generated. A starting point and a velocity are randomly chosen for the track. These are used to specify the initial x and y position and the initial *vel*. Rules 4 through 9 are then used to complete the generation of the track. Each application of a rule also involves a random choice of acceleration, *acc*, that is used in the specification of the problem instance.

The new symbols, each with their own, unique feature list, are added to either a queue, if the symbol is a nonterminal, or an output set, if the symbol is a terminal, and the whole process is repeated. The first element of the queue is removed and processed, and the resulting new symbols are added to the tail of the queue or to the output set. This continues until the queue is empty and the output set contains terminals with fully specified feature lists. More formally, the generation process is defined in Fig. 4.26.

Note that an item's position within the OUTPUT-SET in Fig. 4.26 is insignificant. All ordering constraints must be specified in the feature list. For example, the "time" at which an event occurs is represented as a characteristic of a symbol represented in its feature list.

As discussed previously, the basis of the IDP/*UPC* analysis framework is the assumption that the problem instances of a domain occur in patterned, principled ways. We associate events that might shape the characteristics of a problem instance with rules of a grammar that represent, for example, the occurrence of an event that shifts a signal slightly or that introduces noise or missing data. Thus, the feature list functions associated with the rules of a grammar modify the feature lists appropriately to represent the events associated with the production rules.

For example, one of the characteristics represented in the feature lists of a vehicle tracking domain is "energy." Some of the production rules of the grammar represent events that affect the perceived energy of a signal. These events may increase or decrease this level. Similarly, other characteristics might include "position," "frequency," etc., and they will all be included in the feature list representation and similarly influenced by which production rules are chosen.

In the experimental test domains[7], the credibility generation function for a nonterminal element on the RHS of a production rule is:

Equation 4.6.1 *credibility* - $delta_p$.

[7] The use of a particular credibility generation or interpretation function in an example should *NOT* be considered a restriction on the general applicability of the analysis framework. In fact, the framework can be applied in the analysis of a problem domain or problem solving architecture regardless of the semantic credibility functions used. Selection of different credibility functions will only affect the variance of a domain or problem solver's characteristics. The credibility functions chosen for the examples in this thesis were made to simplify the verification and presentation processes and should not be misinterpreted as limitations on the framework.

INITIALIZE Generation Queue (GQ) - Place the start symbol on GQ.
REPEAT (Until GQ is empty)
  REMOVE Current Grammar Element (CGE) from GQ.
  EXPAND CGE:
    IF CGE is a terminal symbol
      Add CGE to OutputSet
    ELSE
      RANDOMLY CHOOSE a Grammar Rule Corresponding to
        one of CGE's Right-Hand-Sides (RHS).
      FOR each element of RHS
        Set feature list equal to $L_p(CGE_{fl})$
          where L_p is a function that creates a
          new feature list and CGE_{fl} is the
          feature list associated with CGE.
        ADD RHS elements to GQ.
  END REPEAT

Figure 4.26. The Grammar-Based Problem Generation Process

Where "credibility" is the credibility from the feature list of the element on the LHS of the production rule and $delta_p$ is an offset that is used to represent the decrease in credibility associated with noise and missing data. For non-noise/missing data rules, $delta_p$ is 0. For rules that represent the addition of noise or missing data, $delta_p$ is greater than 0. For the experiments in this thesis, $delta_p$ is typically set to 0.2. This is somewhat unrealistic in the sense that "real world" credibility functions would return specific values that might vary considerably from rule to rule. However, this technique does achieve the desired effect of reducing the credibility associated noise and missing data.

The credibility generation function for a terminal element on the RHS of a production rule is:

Equation 4.6.2 *random variable with density N (credibility, variance).*

Where N is the standard normal density, and "credibility" and "variance" are the credibility and "variance" from the feature list of the element on the LHS of the production rule. "Credibility" corresponds to the expected value of the normal density and "variance" corresponds to the variance.

Intuitively, the credibility of the start symbol is passed down through the generation process to the terminal symbols that constitute the actual signal that is input to the problem solver. As the credibility is passed down from feature list to feature list, it is modified only to reflect effects associated with noise and missing data. In general, we associate noise and missing data with domain events that reduce the credibility of an interpretation, and the feature list generation functions associated with noise and missing data rules reflect this by reducing the credibility included in the new feature lists. Finally, the credibility assigned to terminal symbols is a random number generated with an expected value and variance equal to the credibility and variance passed to the feature list of the terminal symbol.

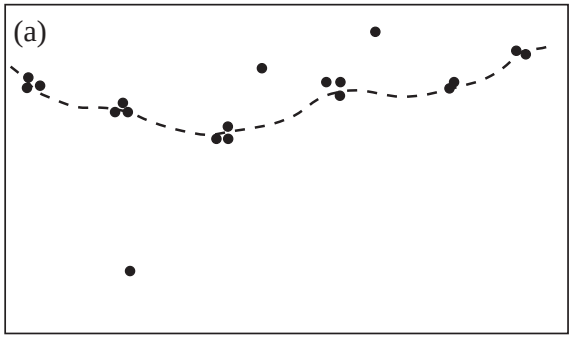
For clarification, the notation "p=0.45" following a production rule should be interpreted to mean, "If the grammar rule associated with this Right-Hand-Side (RHS) is invoked to generate a problem instance, the probability that this specific RHS will be used to expand the rule is equal to 0.45." Note that many of the rules have p=1.0. This indicates that there are no alternative RHSs for this rule. The probabilities associated with a production rule sum to 1. Also, many rules include a RHS of λ. This indicates that there is a possibility that the production rule will not lead to the production of signal data. This is discussed below.

The rules in Fig. 4.25 are used to generate the high-level track phenomena. The recursive rule number 2 will generate a number of track phenomena for each scenario. By adjusting the probabilities for the two alternative RHSs, this rule can be tailored to generate any number of tracks. If the probability of the first RHS is close to 1, this rule will generate many tracks. If it is close to 0, this rule will generate only one track.

Simple scenario examples are shown in Fig. 4.28. In all these examples, the tracks shown move from left to right. This is a convention used for presentation clarity only. The actual grammar can generate tracks that originate anywhere on the sensed region's perimeter and at any point in time. Thus, if an experimental run is to simulate a time span of several days, the vehicle tracks that are generated can begin at any point in time. A single track is shown in Fig. 4.28.a. Figure 4.28.b is an example of multiple tracks in a single scenario. A pattern track is shown in Fig. 4.28.c. This is contrasted with the ghost track in Fig. 4.28.d. Notice that the ghost track differs from the pattern track in that the data is more "spotty." There is more missing data and more time instances for which data is altogether lacking.

23.	$G1_{[f,t,x,y]}$	$\rightarrow S1_{[f,t,x,y]}\ S2_{[f,t,x,y]}$	p=0.45
		$\rightarrow S1_{[f,t,x,y]}\ S3_{[f,t,x,y]}$	p=0.1
		$\rightarrow S1_{[f,t,x,y]}\ S4_{[f,t,x,y]}$	p=0.1
		$\rightarrow S2_{[f,t,x,y]}\ S3_{[f,t,x,y]}$	p=0.1
		$\rightarrow S2_{[f,t,x,y]}\ S3_{[f,t,x,y]}\ S4_{[f,t,x,y]}$	p=0.1
		$\rightarrow S1_{[f,t,x,y]}$	p=0.05
		$\rightarrow S2_{[f,t,x,y]}$	p=0.05
		$\rightarrow \lambda$	p=0.05
24.	$G3_{[f,t,x,y]}$	$\rightarrow S5_{[f,t,x,y]}\ S7_{[f,t,x,y]}$	p=0.45
		$\rightarrow S5_{[f,t,x,y]}\ S6_{[f,t,x,y]}$	p=0.1
		$\rightarrow S6_{[f,t,x,y]}\ S7_{[f,t,x,y]}$	p=0.1
		$\rightarrow S4_{[f,t,x,y]}\ S5_{[f,t,x,y]}$	p=0.1
		$\rightarrow S7_{[f,t,x,y]}\ S8_{[f,t,x,y]}$	p=0.1
		$\rightarrow S5_{[f,t,x,y]}$	p=0.05
		$\rightarrow S7_{[f,t,x,y]}$	p=0.05
		$\rightarrow \lambda$	p=0.05
25.	$G7_{[f,t,x,y]}$	$\rightarrow S11_{[f,t,x,y]}\ S15_{[f,t,x,y]}$	p=0.55
		$\rightarrow S11_{[f,t,x,y]}\ S16_{[f,t,x,y]}$	p=0.43
		$\rightarrow \lambda$	p=0.02
26.	$G8_{[f,t,x,y]}$	$\rightarrow S13_{[f,t,x,y]}\ S18_{[f,t,x,y]}$	p=0.55
		$\rightarrow S13_{[f,t,x,y]}\ S17_{[f,t,x,y]}$	p=0.1
		$\rightarrow S14_{[f,t,x,y]}\ S18_{[f,t,x,y]}$	p=0.1
		$\rightarrow S15_{[f,t,x,y]}\ S17_{[f,t,x,y]}$	p=0.1
		$\rightarrow S13_{[f,t,x,y]}$	p=0.05
		$\rightarrow S18_{[f,t,x,y]}$	p=0.05
		$\rightarrow \lambda$	p=0.05
27.	$G12_{[f,t,x,y]}$	$\rightarrow S6_{[f,t,x,y]}\ S14_{[f,t,x,y]}\ S17_{[f,t,x,y]}$	p=0.45
		$\rightarrow S6_{[f,t,x,y]}\ S14_{[f,t,x,y]}$	p=0.25
		$\rightarrow S7_{[f,t,x,y]}\ S14_{[f,t,x,y]}\ S18_{[f,t,x,y]}$	p=0.25
		$\rightarrow \lambda$	p=0.05

Figure 4.27. Grammar Rules for Generating Group and Signal Data



● : signal data from sensor. Each point has a time, location, energy level, and frequency.

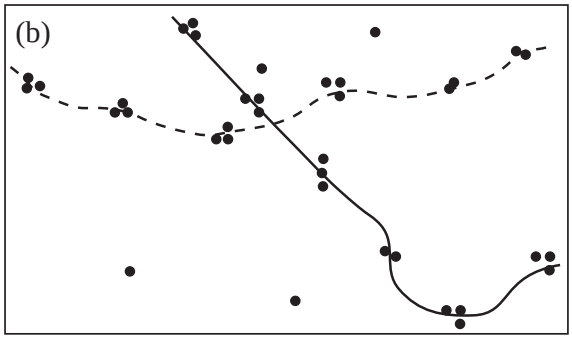
a. single vehicle track

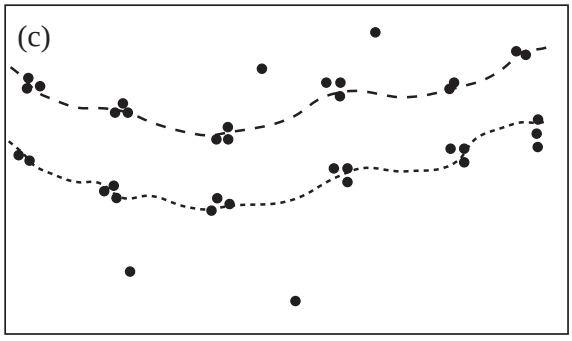
: the actual path of the vehicle that generated the data (a).

b. multiple vehicle tracks

: the actual path of the first vehicle.

: the actual path of a second vehicle.

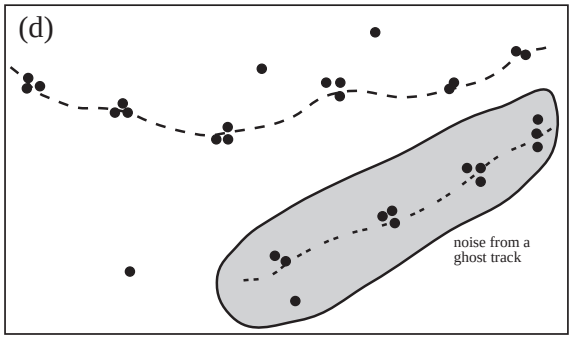




c. patterned vehicle tracks

: the actual path of the first vehicle in the pattern.

: the actual path of the second vehicle in the pattern.



d. single vehicle track with a ghost track

: the actual path of the vehicle that generated the data.

: a ghost image of the actual path.

Figure 4.28. Vehicle Tracking Scenario Examples

Production rule 3 illustrates an interesting principle associated with grammar based analysis. This rule has 6 RHSs, each corresponding to a different vehicle type. For example, the rule "Track $\rightarrow$ I-Track1" indicates that the track is generated by a vehicle of type 1. Similarly, the rule "Track $\rightarrow$ I-Track2" indicates that the track is generated by a vehicle of type 2. The information indicating which type of vehicle caused a track could easily be incorporated into the feature list. This would be done by replacing all 6 RHSs with the single rule, "Track $\rightarrow$ I-Track." However, this would eliminate a considerable amount of power from the analysis tools built from the grammar. However, a general principle that we have recognized is that the power of the analysis tools is a function of the extent to which the IDP grammar explicitly represents domain structures.

It is still possible to analyze structures of domain properties associated with feature lists (for example, credibility values derived from the energy level of a sensed signal), but the analysis results are often less definitive. Specifically, the credibility structure of a domain can be thought of as a set of mean values and variances from those values associated with the set of terminals and nonterminals of an IDP grammar. In certain instances, relationships affecting the credibility rating of a grammar element can be extracted and represented explicitly, as is done with the concept of singularities in Chapter 5.1.1. In so doing, the analysis results associated with this relationship can be extremely precise (e.g., the relationship is true 50% of the time, or the constraint associated with this relationship eliminates certain related search paths x% of the time, etc.), as opposed to the less precise mean and variance estimates. In addition, explicitly representing domain structures can reduce the variance from an expected mean.

4.7 Representing Real-World, Complex Domains

There is no viable method available that can be used to prove that context-free grammars, extended with the feature list convention, can be used to model *all* real-world domains. Instead, we will demonstrate the effectiveness of this approach by specifying a number of real-world phenomena that characterize the interpretation domain we are studying and showing that the phenomena can be modeled accurately with the IDP formalism. The examples presented here will allow the reader to imagine similar techniques being used to model phenomena in other real-world domains.

4.7.1 Interacting Phenomena

A similar feature list convention is used to generate pattern and ghost tracks. This is shown in rules 6 through 9 and rules 10 through 13. These rules lead to domain events consisting of two tracks with closely related properties. As shown in Figures 4.28.c and d, ghost tracks and pattern tracks move in coordination with each other. Ghost tracks do this because one signal is a reflection of a true signal and pattern tracks do this because they are composed of multiple vehicles moving according to a plan. For example, a pattern track might include a tanker vehicle and a second vehicle following it while refueling, multiple vehicles moving in an attack pattern, etc.

Rules 6 through 9 generate the data for each time-frame of the tracks. Rules 10 through 13 generate the patterned data. The vehicle patterns are generated using feature lists and offsets. Each element in a track, which will be referred to as a vehicle location, has an x and a y coordinate. As seen in rule 6, each element of a track has a basic x and y coordinate, but one element has a position modified by a variable *offset*. The offset can be determined by an

28.	G-G1$_{[f,t,x,y]}$	$\rightarrow$ S1$_{[f,t,x,y]}$ S2$_{[f,t,x,y]}$	p=0.2
		$\rightarrow$ S1$_{[f,t,x,y]}$ S3$_{[f,t,x,y]}$	p=0.05
		$\rightarrow$ S1$_{[f,t,x,y]}$ S4$_{[f,t,x,y]}$	p=0.05
		$\rightarrow$ S2$_{[f,t,x,y]}$ S3$_{[f,t,x,y]}$	p=0.05
		$\rightarrow$ S2$_{[f,t,x,y]}$ S3$_{[f,t,x,y]}$ S4$_{[f,t,x,y]}$	p=0.05
		$\rightarrow$ S1$_{[f,t,x,y]}$	p=0.2
		$\rightarrow$ S2$_{[f,t,x,y]}$	p=0.2
		$\rightarrow \lambda$	p=0.2
29.	G-G3$_{[f,t,x,y]}$	$\rightarrow$ S5$_{[f,t,x,y]}$ S7$_{[f,t,x,y]}$	p=0.2
		$\rightarrow$ S5$_{[f,t,x,y]}$ S6$_{[f,t,x,y]}$	p=0.05
		$\rightarrow$ S6$_{[f,t,x,y]}$ S7$_{[f,t,x,y]}$	p=0.05
		$\rightarrow$ S4$_{[f,t,x,y]}$ S5$_{[f,t,x,y]}$	p=0.05
		$\rightarrow$ S7$_{[f,t,x,y]}$ S8$_{[f,t,x,y]}$	p=0.05
		$\rightarrow$ S5$_{[f,t,x,y]}$	p=0.2
		$\rightarrow$ S7$_{[f,t,x,y]}$	p=0.15
		$\rightarrow \lambda$	p=0.25
30.	G-G7$_{[f,t,x,y]}$	$\rightarrow$ S11$_{[f,t,x,y]}$ S15$_{[f,t,x,y]}$	p=0.30
		$\rightarrow$ S11$_{[f,t,x,y]}$ S16$_{[f,t,x,y]}$	p=0.30
		$\rightarrow \lambda$	p=0.40
32.	G-G8$_{[f,t,x,y]}$	$\rightarrow$ S13$_{[f,t,x,y]}$ S18$_{[f,t,x,y]}$	p=0.15
		$\rightarrow$ S13$_{[f,t,x,y]}$ S17$_{[f,t,x,y]}$	p=0.05
		$\rightarrow$ S14$_{[f,t,x,y]}$ S18$_{[f,t,x,y]}$	p=0.05
		$\rightarrow$ S15$_{[f,t,x,y]}$ S17$_{[f,t,x,y]}$	p=0.05
		$\rightarrow$ S13$_{[f,t,x,y]}$	p=0.2
		$\rightarrow$ S18$_{[f,t,x,y]}$	p=0.25
		$\rightarrow \lambda$	p=0.25
32.	G-G12$_{[f,t,x,y]}$	$\rightarrow$ S6$_{[f,t,x,y]}$ S14$_{[f,t,x,y]}$ S17$_{[f,t,x,y]}$	p=0.2
		$\rightarrow$ S6$_{[f,t,x,y]}$ S14$_{[f,t,x,y]}$	p=0.2
		$\rightarrow$ S7$_{[f,t,x,y]}$ S14$_{[f,t,x,y]}$ S18$_{[f,t,x,y]}$	p=0.25
		$\rightarrow \lambda$	p=0.35

Figure 4.29. Grammar Rules for Generating Group Data for Ghost Tracks

arbitrarily complex function, resulting in a very expressive technique for generating coordinated domain events.

This is just one example of how the feature list convention can be used to generate domain phenomena that are coordinated or related in some way. This is worth noting because it suggests that the analysis techniques we will present here can be extended to many other domains. Even domains that seem to have some context-sensitive phenomena that appear to interact through time and/or distance. This is a very important point because it suggests that, in some situations, it is possible to use a context-free grammar to generate problem instances that can be considered context-sensitive. The significance of this should not be understated. The power of the analysis tools presented here is derived in large part from their context-free nature. It is not clear at this time that the validity of the analysis tools will hold for context-sensitive grammars. Nor is it clear that some of the analysis results can even be computed for domains represented by context-sensitive grammars.

4.7.2 Noise

In any interpretation domain, noise plays a significant role in increasing the complexity of problem solving. This is the result of noise increasing the number of plausible interpretations (i.e., increasing the ambiguity) that have to be differentiated. A variety of techniques have been developed for modeling different types of noise.

Production rules 18, shown in Fig. 4.25, and 35, shown in Fig. 4.30, are used to generate *random noise*. This represents random phenomena associated with natural state of the domain when no vehicles are passing through the region. There could be many causes of random noise including temperature changes, equipment malfunctions, natural flora or fauna, and more. The amount of random noise in a domain is determined by the probabilities associated with rule 18. If the probability of the first RHS is high, the domain will include a great deal of noise. If the probability is low, very little noise will be generated. The properties of the random noise can be further modified by adjusting the probabilities associated with the RHSs of rule 35.

In addition to adjustments to the probability distributions, random noise characteristics can be modified with the feature list convention. For example, note that the feature lists used in rule 18 do not include any information about x and y locations. In the vehicle monitoring system, the x and y locations of random noise are determined using a uniform distribution function. This does not have to be the case. It would be easy to model a domain in which random noise tended to appear more in certain locations. Other properties of random noise, such as its energy level, can be manipulated in a similar way.

In addition to random noise, there is often noise that is more closely associated with the domain events that the system is trying to interpret. For example, in a vehicle tracking domain, it is reasonable to expect there to be phenomena corresponding to a vehicle interacting with the domain in an unexpected way. For example, a train may hit a rock or some other object on the track, something may fall off a ship, or a jet engine may emit some uncharacteristic noise. These sorts of phenomena are different from random noise, since they will only occur in the presence of a vehicle, but they are still a form of noise since they are not characteristic of a vehicle and may lead to ambiguity or otherwise obscure the sensing of phenomena that is characteristic of a vehicle. Rule 23 shows an example of how this sort of noise can be represented by adding elements to the RHS of an existing rule. Specifically, in the rule,
$G1_{[f,t,x,y]} \rightarrow S2_{[f,t,x,y]}\ S3_{[f,t,x,y]}\ S4_{[f,t,x,y]}$, the extra signal, S4, could be considered noise.

$$
\begin{array}{lll}
35. \quad n_{[f,t]} & \rightarrow S1_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S2_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S3_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S4_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S5_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S6_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S7_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S8_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S9_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S10_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S11_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S12_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S13_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S14_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S15_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S16_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S17_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S18_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S19_{[f,t,x,y]} & p=0.05 \\
& \rightarrow S20_{[f,t,x,y]} & p=0.05
\end{array}
$$

Figure 4.30. Grammar Rules for Generating Random Noise

The representation of noise does not have to be restricted to the signal level phenomena. More comprehensive noise elements can be represented at other levels of the grammar in a similar way. For example, ghosting phenomena are really a complex aggregate of noise at the track level. Noise phenomena can be easily added at the group level in a manner similar to that used to add signal data noise.

Another kind of noise is represented not by additional phenomena, but by slightly altered phenomena. This is sometimes referred to as sensor shifting. Intuitively, sensor shifting phenomena can be caused by errors in sensors or variations in a physical domain such as air or water temperature. All of the group level production rules include examples of sensor shifting. For example, the second RHS of rule 23 shows a shift from S2 to S3. Similar shifting can occur at all levels of the grammar.

4.7.3 Correlated and Uncorrelated Noise

In Chapter 4.1.2, we identified two kinds of noise, Correlated and Uncorrelated. Correlated noise leads to the generation of additional interpretations. This ambiguity causes the problem solver to perform additional work to differentiate the possible interpretations. Uncorrelated noise may lead to additional work, but it does not lead to the generation of additional interpretations. Often, uncorrelated noise can be identified and eliminated from further processing with low-cost.

The complexity of the example grammar is such that it is difficult to identify all correlated and uncorrelated noise by inspection only. However, the rules associated with the nonterminal

n include examples of both. For example, production rule 35.$n_{[f,t]} \rightarrow S1_{[f,t,x,y]}$ is an example of correlated noise because the signal S1 can be used to build an interpretation of an *I-Track1* in situations where it would not otherwise be built. In contrast, the production rule 35.$n_{[f,t]} \rightarrow S20_{[f,t,x,y]}$ is an example of uncorrelated noise because the signal data S20 is not used to generate any vehicle tracks and can be ignored in subsequent processing.

A more meaningful example of correlated noise is shown in the problem instance illustrated in Fig. 4.31. In this example, the production rule $N_{[f,t]} \rightarrow n_{[f,t]}\ N_{[f,t]}$ will create random noise throughout the sensed region. Should some of this noise occur near a track, it will cause the problem solver to create a possible interpretation that includes the noise as a possible element of the track. This interpretation could be in conflict with the interpretation that corresponds most closely with the event that created the noise. This is shown in the figure with the two different "track lines." In this instance, the random noise phenomenon is considered correlated noise.

Also shown in the figure is random noise that is too far from the actual track data to be combined into a competing interpretation. This random noise phenomenon is considered uncorrelated noise.

4.7.4 Missing Data

Missing data, defined in Chapter 4.1.3 is easy to represent and understand using IDP grammars. By simply dropping an element from an RHS, it is possible to model phenomena that results in data not being sensed. Missing data can be caused by sensor errors, environmental conditions, processing errors, and more. The second and third RHSs to rule 19 show examples of how missing group level data can be modeled by omitting the appropriate nonterminal. The sixth and seventh RHSs of rule 23 show how missing signal data can be modeled in a similar way.

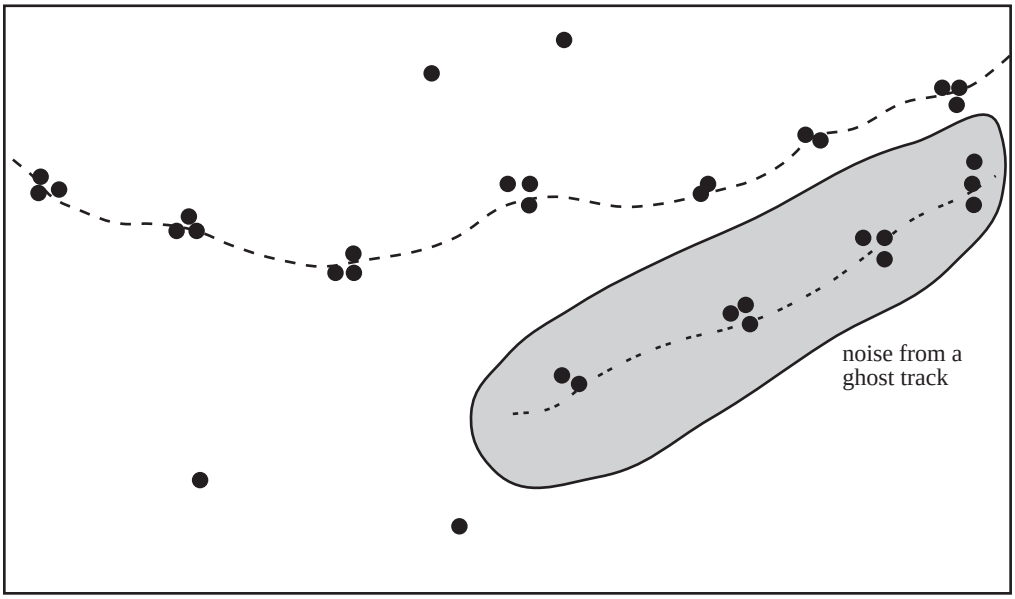
Another mechanism for representing missing data is in the form of λ rules. These are production rules that transform nonterminals to the empty set. These RHSs are used to model domain phenomena such as entirely missing groups of related data. For example, in real-world domains, missing vehicle locations in a track are not unheard of. This is modeled with the production rule V1 $\rightarrow \lambda$. Note that for ghost tracks, the probability of missing vehicle locations is much higher than it is for real vehicle tracks.

The production rules associated with ghost tracks also demonstrate how missing data phenomena can be related across production rules. Note that the RHSs for the ghost group level nonterminals are identical with the RHSs for the regular group level nonterminals. The only difference is in the distribution probabilities. For ghost data, the probability of missing signal data, or completely missing group data, is much higher. This is one of the ways in which ghost tracks can be identified. In this example, the related domain events, i.e., corresponding levels of missing data, are linked not by the feature list convention, but by a common nonterminal.

4.8 An Example of Basic Analysis Using IDP Models

This section will demonstrate how IDP models can be used to analyze sophisticated control mechanisms. In particular, we will focus on the use of abstractions and approximations[8] in

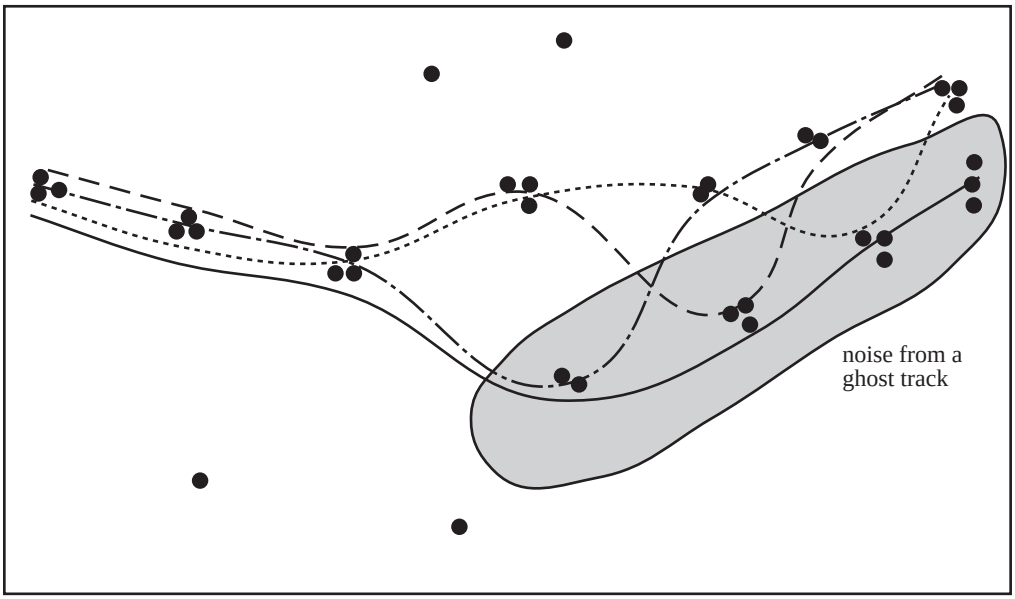
[8] In the remainder of this thesis, both abstractions and approximations will be referred to simply as abstractions except in cases where it is necessary to differentiate them.



● : signal data from sensor. Each point has a time, location, energy level, and frequency.

: the actual path of the vehicle that generated the data.

: a ghost image of the actual path.



ambigous tracks derived from the correlated noise of the ghost track

Figure 4.31. Example of Correlated Noise in a Vehicle Tracking Domain

sophisticated control architectures. The examples presented will help motivate the development of the *UPC* formalism that will be introduced in Chapter 6.

The analytical power of an IDP domain model is based on the degree to which the abstractions used in control can be viewed from the same perspective as problem solving actions. Thus, the key to analyzing abstractions is to view them in terms of the domain problem structures they define. In the IDP formalism, a domain's problem solving actions are represented in terms of the domain's characteristic grammar and the associated functions. These operators generate fully specified partial interpretations and ignore all considerations of the efficiency of their actions. This is in contrast to meta-operators based on abstractions or approximations that generate partial interpretations that are underconstrained in some way. For example, a characteristic variable may be undefined or may be defined by a range of values. Therefore, analyzing abstractions used in sophisticated control mechanisms within the context of a given domain requires that the domain's grammar and associated functions be modified to represent the abstractions as problem solving actions. The modified grammar then defines a new convergent search space and comparative analysis can focus on the relative efficiency of problem solving in the original and the modified search spaces.

Once a domain's grammar has been modified to represent available abstract problem solving actions, the subsequent analysis must focus on two issues. The first will be referred to as *correctness*. Analysis must demonstrate that the problem solving actions represented in the modified grammar generate the same (or acceptably different) results as the problem solving actions represented in the original grammar. The second issue is *efficiency*. Analysis must demonstrate that problem solving in the search space defined by the modified grammar is more efficient than problem solving in the search space defined by the original grammar. Note that this methodology implies that abstract problem solving operators are used solely to improve a problem solver's efficiency.

Proof of efficiency can take a number of forms depending on whether top-down or bottom-up (or both) methods are used. In both cases, analysis will focus on the expected costs to connect the search spaces both before and after modifications corresponding to abstract operators are made. If bottom-up methods are being employed, the analysis must start with the low-level components and show that the cost of generating interpretations from these elements has been decreased. If top-down methods are used, the analysis must start with the high-level representation of the set of interpretations and demonstrate that the cost of pruning this set is somehow decreased. As will be seen, both top-down and bottom-up problem solving can be analyzed similarly when viewed from the perspective of states in a search space.

The analysis contained in this section will focus on abstractions used in sophisticated control mechanisms that have been implemented in the *Distributed Vehicle Monitoring Testbed (DVMT)* [Corkill, 1983] and that have been implemented to exploit problem structures of non-monotonic domains. Such domains are characteristic of real-world domains such as signal interpretation, robotic audition, image processing, and natural language processing.

4.8.1 Goal Processing

Goal processing is a form of hierarchical problem solving [Knoblock, 1991a] that has been incorporated in the DVMT to enable a problem solver to reason about courses of action in ways that are independent of the means for instantiating the actions. After new search states are generated, partial constraints are applied to the states to generate meta-states, or *goals*. Further problem solving actions are then applied to the goals and the original search states

are considered to be connected and are no longer used to initiate problem solving activity. In numerous studies, goal processing has been shown to be an effective means for countering local redundancy and uncertainty [Corkill and Lesser, 1981, Corkill *et al.*, 1982, Corkill, 1983]. In addition, goal processing has been shown to be effective in top-down processing algorithms where goals are used to constrain the actions of data-directed operators [Corkill and Lesser, 1981, Corkill *et al.*, 1982, Corkill, 1983].

Intuitively, goals can be thought of as "set descriptors." A goal is similar to a state of the original search space, but it is underconstrained in the sense that some of the characteristic variables that normally define states in the search space are unspecified or specified in terms of a range of values. As a consequence, any state with characteristics that fall within the ranges defined by the goal can be thought of as elements of a set represented by the goal. Hence, goals are abstractions of search space states. In the initial DVMT implementation, goals define a set that represents the elements of a state's result set (defined in Chapter 4.4.1) that can be generated with the application of a single operator. Extending a goal state can be thought of as a branch-and-bound search operation applied to the goal.

Returning to grammar G' from Fig. 3.2, we see that the structure of the search space defined by this grammar is very simple. In fact, problem solving in this domain would be more appropriately thought of as classification problem solving since it would be trivial to preenumerate the set S from which an interpretation would be chosen. Furthermore, there is no ambiguity, so interpretations could be determined without using the evaluation functions f_p. Therefore, to make the following examples more meaningful, the extended grammar, G'_n, shown in Fig. 4.32, will be used.

Figure 4.32.a represents the operator organizational structure for the example domain problem. Each of the numbered macro-operators shown in Fig. 4.32.a is actually a set of primitive rules – a "best rule," which is listed first, and subsets of masking rules and missing data rules. For example, rule 1 specifies the production $A_n \rightarrow C_n D_n$ and the eight associated masking rules specified by the subscript $n \pm 1$. (n might correspond to a slight variation in position, time, frequency, etc.) Thus, $A_n \rightarrow C_{n-1}D_{n-1} \mid C_{n-1}D_n \mid C_{n-1}D_{n+1} \mid \ldots$, are all primitive operators that will be applied when macro-operator 1 is applied. In addition, some of the macro-operators include primitive missing data rules. These are macro-operators 3 through 6. The primitive missing data rules are 3.b&c, 4.b&c, 5.b&c, and 6.b through g. For now, the evaluation, cost, and distribution functions will be left undefined.

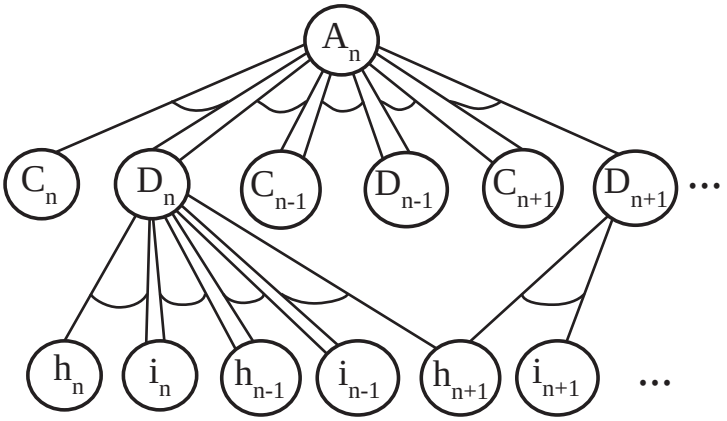
Figures 4.32.b and 4.32.c illustrate structures of the grammar in tree form. Figure 4.32.b depicts structures associated with the masking rules and Fig. 4.32.c shows structures associated with missing data rules. Note that this grammar is very ambiguous – any input can lead to the derivation of many different interpretation trees. For example, an input of "$f_2 g_2 h_2 i_2$" will result in the generation of several dozen full interpretation trees (see Fig. 4.33.). Depending on the semantics of the domain, each of these interpretations may be virtually identical or vastly different.

Figures 4.34 and 4.35 illustrate the effects of goal processing. These figures are based on the extended interpretation grammar G'_n from Fig. 4.32. Figure 4.34 depicts a typical interpretation search for a small set of input data[9]. To connect the search space, op_6 is successively applied to states x_1, y_1, y_2, and z_1. These operations result in the creation of states W_0, W_1, W_2 and W_3. As shown in the figure, W_0, W_1 and W_2 are actually created multiple

[9] In these figures, and in the following text, the operator superscript notation represents the i^{th} application of an operator.

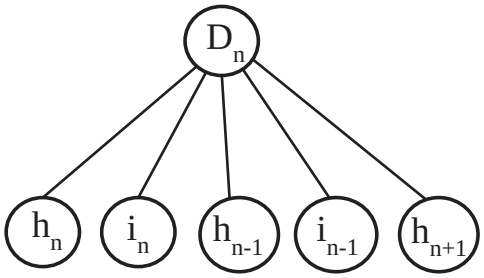
1. $A_n \rightarrow C_{n\pm1}D_{n\pm1}$
2. $B_n \rightarrow D_{n\pm1}E_{n\pm1}W_{n\pm1}$
3.a. $C_n \rightarrow f_{n\pm1}g_{n\pm1}$
3.b. $C_n \rightarrow f_{n\pm1}$
3.c. $C_n \rightarrow g_{n\pm1}$
4.a. $D_n \rightarrow h_{n\pm1}i_{n\pm1}$
4.b. $D_n \rightarrow h_{n\pm1}$
4.c. $D_n \rightarrow i_{n\pm1}$
5.a. $E_n \rightarrow k_{n\pm1}j_{n\pm1}$
5.b. $E_n \rightarrow k_{n\pm1}$
5.c. $E_n \rightarrow j_{n\pm1}$
6.a. $W_n \rightarrow x_{n\pm1}y_{n\pm1}z_{n\pm1}$
6.b. $W_n \rightarrow x_{n\pm1}y_{n\pm1}$
6.c. $W_n \rightarrow x_{n\pm1}z_{n\pm1}$
6.d. $W_n \rightarrow y_{n\pm1}z_{n\pm1}$
6.e. $W_n \rightarrow x_{n\pm1}$
6.f. $W_n \rightarrow y_{n\pm1}$
6.g. $W_n \rightarrow z_{n\pm1}$

a. extended interpretation grammar, G_n'



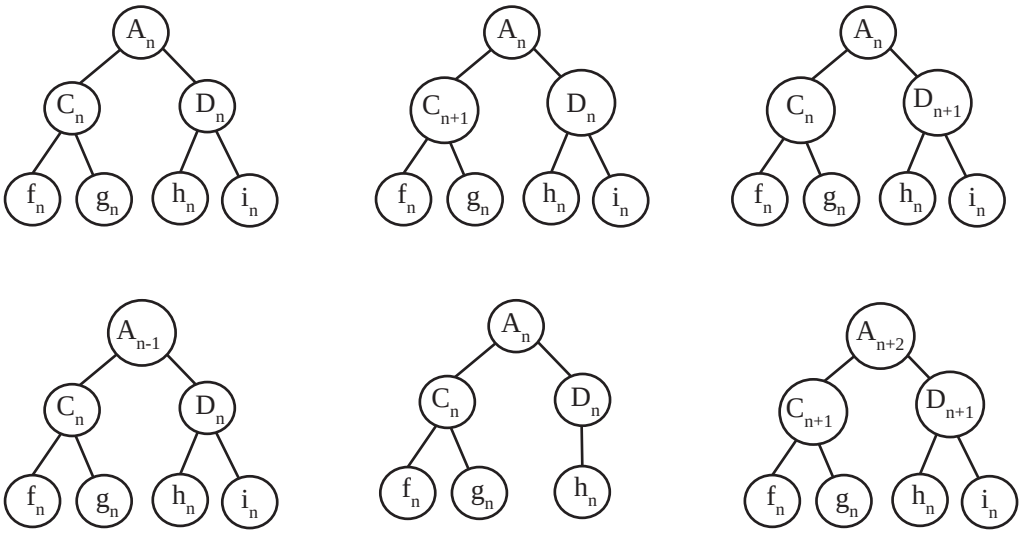
(signal data)

b. example structures from grammar



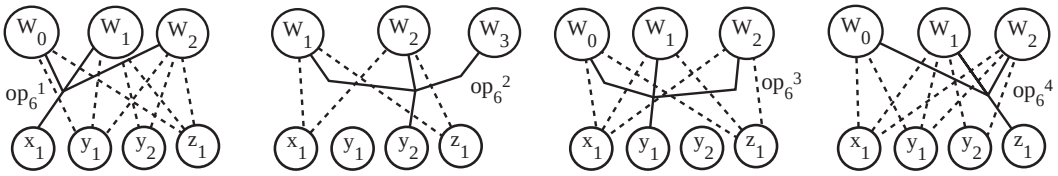
c. additional intermediate structures from grammar

Figure 4.32. Extended Interpretation Grammar



Some of the interpretation trees resulting from the input "$f_2g_2h_2i_2$."

Figure 4.33. Example of Interpretations Based on Extended Grammar G'_n



a. operator 6 applied to x_1

b. operator 6 applied to y_2

c. operator 6 applied to y_1

d. operator 6 applied to z_1

(Solid lines represent search paths resulting from operator application and dashed lines represent implied merge operations with supporting data used by the operators.)

Figure 4.34. Example of Interpretation Search

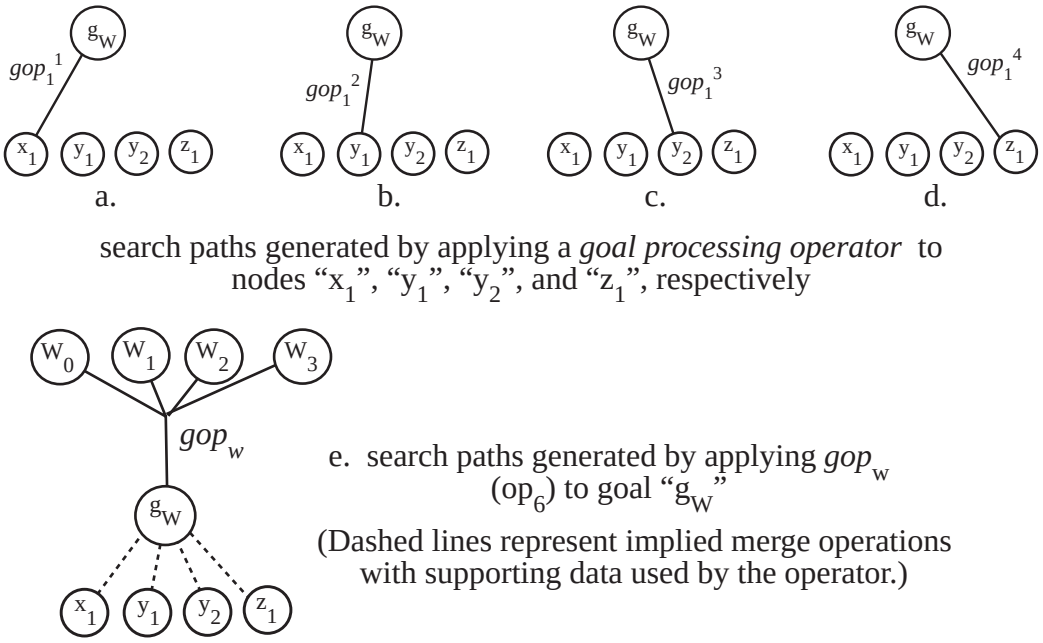


Figure 4.35. Example of Interpretation Search Using Goal Processing

times. The dashed lines indicate the information used by the operator in each of the search steps. For example, when op_6 is applied to x_1, the operator uses the states y_1, y_2, and z_1 in its processing.

Figure 4.35 is an example of how goal processing functions in the same situation. Instead of op_6, a goal processing operator, gop_1, is applied to x_1, y_1, y_2, and z_1, in each case creating a new *goal state*, g_W, and connecting the original search states. g_W can be thought of as an abstract state that represents the problem solver's intention to extend states x_1, y_1, y_2, and z_1. Because of the characteristics of x_1, y_1, y_2, and z_1, this intention is similar for each and can be represented as a single goal state. Another goal processing operator, gop_W, is then applied to g_W. In this situation, gop_W can be a slightly modified op_6, as shown in Fig. 4.35.e, that combines op_6^1, op_6^2, op_6^3, and op_6^4 in a single operator. The application of gop_W to g_W results in the generation of W_0, W_1, W_2 and W_3. Again, the dashed lines represent the information used by gop_W.

This form of goal processing has several advantages. It may require significantly less work than the interpretation search process discussed above and shown in Fig. 4.34. For example, the individual search operations op_6^1, op_6^2, op_6^3, and op_6^4 might have a much higher fixed overhead cost than gop_W. Also, the individual search operations must redundantly search the database for inputs and many of the results that are produced are also redundant. It may be possible to avoid some of these costs by using a single goal operator.

Furthermore, this form of goal processing allows a problem solver to reason about the goals themselves. Corkill and Lesser discuss the advantages of this capability in [Corkill and Lesser, 1981, Corkill *et al.*, 1982, Corkill, 1983], and this work is further extended in [Lesser *et al.*, 1989a, Lesser *et al.*, 1989b, Decker *et al.*, 1990]

Clearly, there are cases where this form of goal processing is not advantageous. For example, in situations where the individual search operations are not redundant or where the overhead

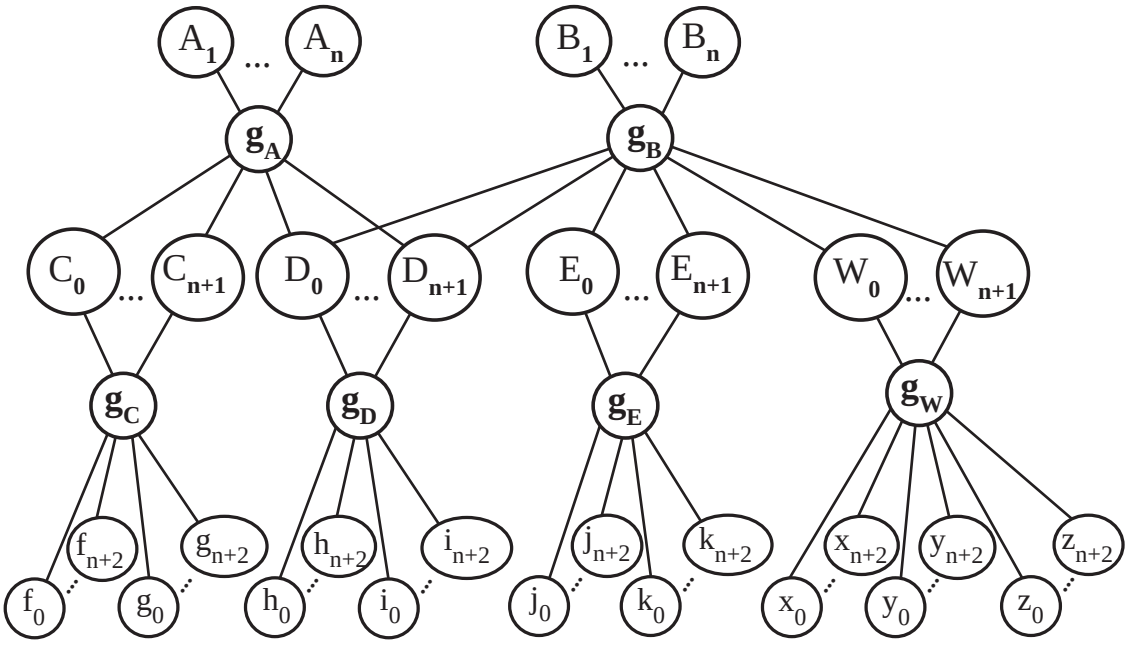


Figure 4.36. Representing Goal Processing in a Grammar

of processing a goal is greater than the savings. Similarly, if the cost of an operation increases exponentially with the size of the input set, it may be better to implement a solution composed of numerous distinct search tasks where the cardinality of the input sets is limited.

The use of goal processing can be represented in the IDP framework by altering G'_n as shown in Fig. 4.36. In the transformed grammar, production rules are added using the meta-states g_A, g_B, etc. Each of these goal states is defined in terms of the result sets of the goals' children states. For example, g_W is the union of the elements of the partial result sets of $x_0 \ldots x_{n+2}, y_0 \ldots y_{n+2}$, and $y_0 \ldots y_{n+2}$ that can be generated with a single operator application. A similar definition is used to specify the other goals.

The operators that are applied to the goal states will be defined as the macro-operators from G'_n defined in Fig. 4.36. Thus, gop_W is applied to goal g_W to generate states $W_0 \ldots W_{n+1}$, gop_E is applied to g_E generate states $E_0 \ldots E_{n+1}$, etc. At the next level of interpretations, gop_A is applied to goal g_A to generate states $A_0 \ldots A_{n+1}$ and gop_B is applied to goal g_B to generate states $B_0 \ldots B_{n+1}$.

Given these definitions, it should be apparent that the use of goal processing does not effect the interpretations that are generated. Since the same operators, with identical inputs, are used to extend the goal states as were used in the original search, the results will be the same. However, the costs will differ. Notice that in the original example, op_6 was applied four times and in the second example with goal processing, gop_W was only applied once, but the goal processing operator was applied four times. Consequently, if the goal processing operator, gop_W, is significantly less expensive than op_6, then goal processing will offer distinct advantages.

Goal processing will be advantageous when the expected cost of connecting the goals plus the cost of generating the goals is less than the cost of connecting the search states without the use of goal processing. The cost of connecting the search states without goal processing is:

Equation 4.8.1 $cost(op_6^1) + cost(op_6^2) + cost(op_6^3) + cost(op_6^4)$.

(For the sake of simplicity, the cost of merging identical states will be ignored.)

Analysis of these costs will use models similar to the cost models from the DVMT [Corkill, 1983]. In the DVMT, the cost of an operator application can be approximated by a constant factor plus a function of the number of inputs and the number of outputs. Specifically:

Equation 4.8.2 $\forall i, cost(op_i, n, m) = con_i + a * n^2 + b * m$,

where con_i is the constant cost of operator i, n is the number of inputs to op_i, m is the number of outputs generated by op_i, and a and b are cost coefficients.

Assuming that constant costs are 1, Equation 4.8.1 yields

$$(1 + 16 + 3) + (1 + 9 + 3) + (1 + 9 + 3) + (1 + 16 + 3) = 66. \quad (4.1)$$

For goal processing, the costs will be:

Equation 4.8.3 $cost(gop_1^1) + cost(gop_1^2) + cost(gop_1^3) + cost(gop_1^4) + cost(op_6^1)$.

Assuming the goal processing operator conforms to Equation 4.8.2 and the constant cost of gop_1 is 1, Equation 4.8.3 yields

$$(1 + 1 + 1) + (1 + 1 + 1) + (1 + 1 + 1) + (1 + 1 + 1) + (1 + 16 + 4) = 33. \quad (4.2)$$

Consequently, in this simple example, goal processing results in a savings of 50%.

Though these figures are approximations, they are representative of the costs of DVMT search operators. The input component, n, of Equation 4.8.2 reflects the cost of retrieving data from the blackboard and the combinatorial nature of the reasoning processes used by DVMT operators [Decker *et al.*, 1990, Corkill, 1983]. The output component, m, reflects the cost of writing data to the blackboard.

In this simple example, goal processing has clear performance advantages. However, it is still unclear as to when, in general, goal processing is effective and when it is detrimental to performance. For example, in the DVMT, an analysis indicated that goal processing is not always an effective tool [Lesser *et al.*, 1989a, Lesser *et al.*, 1989b]. Subsequent work exploited this observation and resulted in significant performance improvements [Decker *et al.*, 1989]. This work was specific to one aspect of goal processing in the DVMT domain, and left open questions regarding the general properties of domains where goal processing is useful. More specifically, this analysis did not consider the potential benefits of the *subgoaling* mechanisms described in[Corkill, 1983].

In the example presented in this section, the principle difference in cost can be attributed to the fact that without goal processing, op_6 had to be applied four times to connect the low-level states. This is necessary because it is impossible to determine a priori whether or not the application of op_6 will result in the generation of a unique interpretation – i.e., an interpretation that will not be generated by any other application of op_6. In this example, the second application of op_6 generates a unique W_3. This is the result of inherent uncertainty in the form of missing data rules. If there were no missing data rules, the structure of the grammar would be such that op_6 would not have to be applied to every state.

A question that arises from this example is whether or not goal processing can be improved by simply applying op_6 to all the low level data simultaneously. This could be accomplished by

creating a new meta-operator that would include all the possible applications of op_6. However, this would limit the problem solver's flexibility by forcing it to always apply op_6 to all the low-level data. This is a viable option for domains that do not offer possibilities for connecting goal states without applying operators to extend them (i.e., domains where it is not possible to prune goal states). In other words, in domains where there is no opportunity for pruning the available operators that would extend a goal, it might be possible to find a control architecture more efficient than goal processing. In the DVMT, operators that extend goals are pruned under certain conditions. Furthermore, such pruning is done often enough that it is advantageous to use goal processing as described in [Corkill, 1983].

4.9 Chapter Summary

This chapter demonstrates how the IDP formalism modifies a domain grammar to represent phenomena such as noise and missing data are discussed. An example of the value of these definitions is demonstrated in Definition 4.1.4, which formally specifies the concept of *correlated noise.* The IDP formalism is also used to specify the concept of *interacting subproblems.* In addition, this chapter shows how sophisticated control mechanisms, such as bounding functions, can be represented in the IDP framework. An extended definition of *monotonicity* is presented in Section 4.5.

A demonstration of how the IDP formalism represents complex, real-world domains, in this case a vehicle tracking domain, is given in Chapter 4.7. This section of the paper demonstrates how the *feature list convention* developed in this thesis can be used to model the characteristics of real-world phenomena. The use of the feature list convention is shown in Chapter 4.6. Finally, Chapter 4.8 demonstrates how the IDP formalism can be used in the analysis of a sophisticated control mechanism, bottom-up goal processing.

CHAPTER 5

QUANTITATIVE ANALYSIS AND EXPERIMENTATION WITH BASIC IDP MODELS

Within the IDP/*UPC* analysis framework, the IDP formalism is used to generate simulated domain events in an experimental testbed and to define a problem solver's control architecture. Furthermore, it forms a basis for the general analysis framework and the specific analysis paradigms discussed in Chapter 1.7. IDP grammar used to generate domain events is referred to as the *domain generation grammar,* IDP_G. The IDP grammar that defines a problem solver will be referred to as the *interpretation grammar,* IDP_I.

The IDP formalism will thus support a variety of experimental activities. Specifically, given a domain, the performance of different control architectures can be tested by modifying the interpretation grammar. Alternatively, given a control architecture, its applicability to and effectiveness in new domains can be measured by altering the domain grammar. Thus, given a control architecture that is very successful in one specific domain, it is possible to identify other domains, or classes of domains, where it will perform equally well. In both these capacities, large numbers of experiments can be run and analyzed quantitatively. In addition, unknown problem structures can be analyzed experimentally using a control architecture as an experimental tool.

Figure 5.1 represents an abstract view of an experimental testbed based on the IDP formalism that has been developed. In this system, a *Domain Simulator* uses the domain grammar to generate signal data corresponding to domain events. A *Problem Solver* with a control component based on a potentially different, *perceived* IDP structural definition (the interpretation grammar) interprets the signal data. The control component used in the experimental framework can be either the statistically optimal control strategy defined in Chapter 6.4, or an arbitrary heuristic control strategy whose operations are defined by IDP$_i$. A detailed description of some of the more important aspects of the functioning of the domain simulator is given in Chapter 4.6.

In addition to its use as an experimental tool, the IDP formalism can also be used as an analytical tool for prediction and explanation. Specifically, the IDP formalism is used to calculate $E(C)$, the expected cost of a single problem solving instance for a given domain. The calculation of $E(C)$ is defined here and its accuracy is verified experimentally in Chapter 7.

Given the experimental paradigm described above, it will be necessary to calculate $E(C)$ for two different scenarios. In one, the domain grammar and the interpretation grammar are identical. In the other, the domain grammar and the interpretation grammar are different. The calculation of $E(C)$ is very similar for both cases. Using the IDP specifications of the domain and interpretation grammars, the *expected frequency,* F_n, of each element (i.e., terminal, nonterminal, SNT) in the interpretation grammar is calculated. F_n represents the number of expected search state instantiations corresponding to grammar element n per problem solving instance. By multiplying F_n by the expected cost of each of the operators that can be applied to

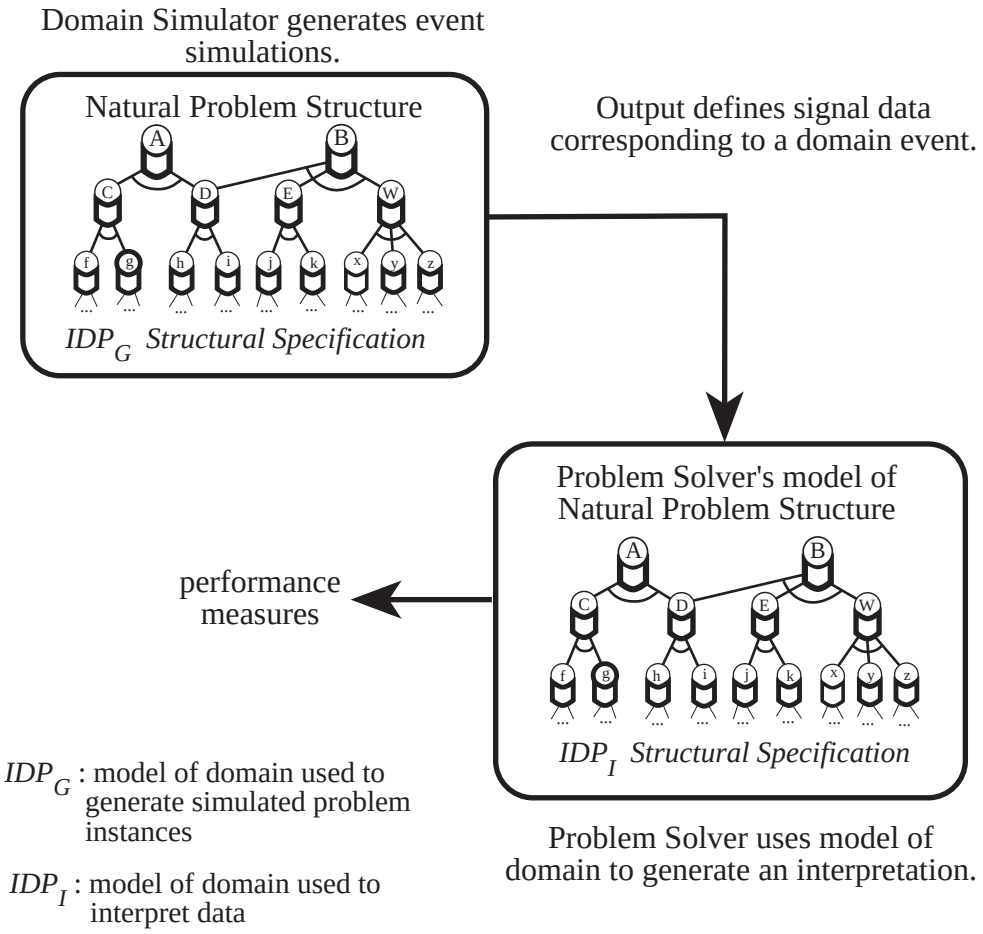


Figure 5.1. Overview of the IDP Model as the Foundation of an Experimental Testbed

n, the expected cost associated with grammar element n is computed. Summing these values for every element of the grammar yields $E(C)$.

It is important to realize that each element of the grammar may have multiple search state instantiations. For example, in a given problem instance, element "A" of the grammar may correspond to many distinct search states. The derivation paths of the distinct states may be differentiated either by different search paths or different inputs, i.e., identical paths with different components. (Two search paths that represent the application of the same sequence of operators will produce different results if any of their inputs are different. In a search space, this is represented by states with slightly different characteristic variables. See Chapter 6 for more details.) If a specific element of the grammar has an expected frequency of five, it means that, on average, five instantiations of that element of the grammar will be made, each instantiation being a unique state in the corresponding search space. These instantiations (or states) will be distinguished by the individual characteristics of the search paths that lead to the creation of the state. Although redundant paths will lead to the same state instantiation representing the same interpretation or partial interpretation, other search paths, though they correspond to the same element of the IDP grammar, represent different interpretations or partial interpretations.

Chapter 6 will explain how IDP structures are represented as characteristics of a search

interpretation trees derived from signal data “fqg rhi”:

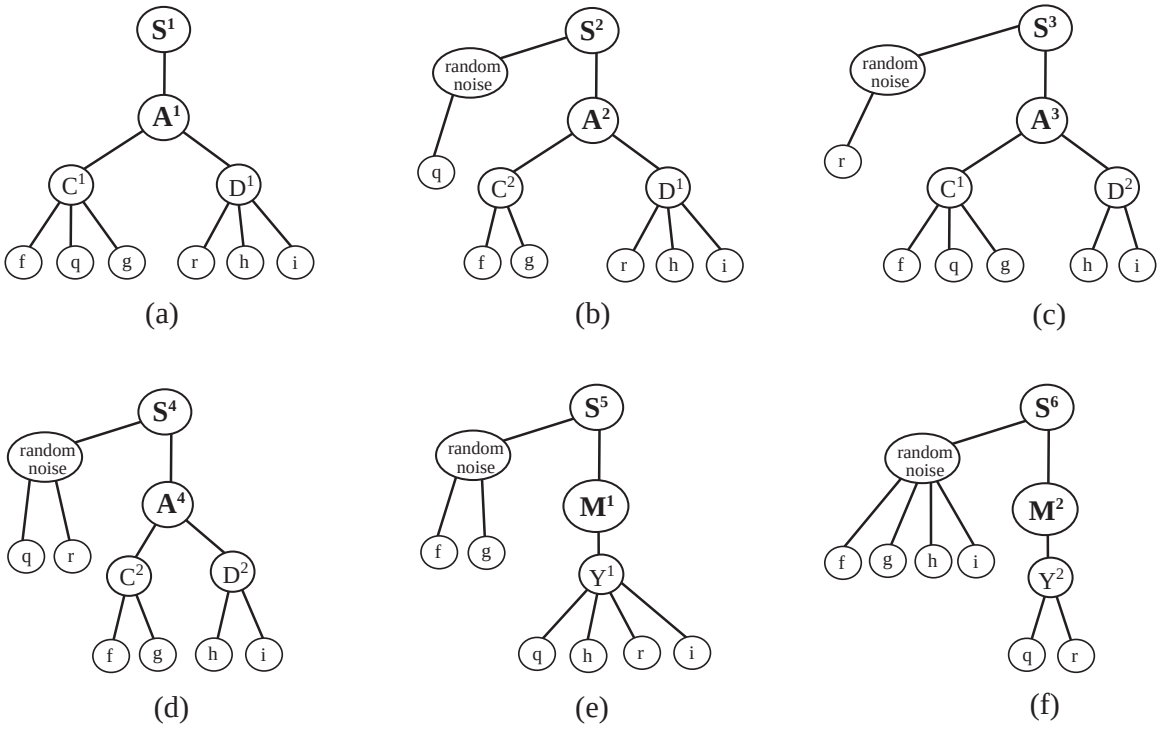


Figure 5.2. Example of Signal Data Leading to Multiple IDP State Instantiations

state’s specification, but it should be intuitive that, for most domains, distinct interpretation trees built from an IDP grammar correspond to distinct search space states. This is because, for the most part, the properties of an interpretation tree are derived from the properties of its subtrees such as “credibility,” “location,” “time,” etc. Chapter 4.6 describes how these properties are generated.

For example, Fig. 5.2 shows a situation where signal data leads to the generation of states representing multiple instantiations of the same IDP element. In Fig. 5.2 the signal data leads to the generation of the six interpretations shown. There are multiple instantiations of the partial result C that are shown in Figs. 5.2.a and 5.2.b as C^1 and in Figs. 5.2.b and Figs. 5.2.d as C^2. The differences between C^1 and C^2 are seen clearly in the figure as differences in the subtrees that were used to generate the specific instantiations of C. (In an actual implementation, this would be implemented as differences in the characteristic variable “supporting data” associated with each of the instantiations of C.) Similarly, there are two distinct instantiations of the partial results Y and D. The different partial result instantiations are used to generate four distinct instantiations of the SNT A and two distinct instantiations of the SNT M. Finally, there are a total of six different interpretations of the data. This is shown as six different instantiations of S.

For element n of the grammar, where n is a nonterminal or SNT, the calculation of F_n is defined in terms of the elements on the RHS of n’s production rules, i.e., the children of n, and any bounding functions incorporated in the grammar. If n is a terminal symbol, its frequency

is based on the function ψ and is derived in a top-down manner from the start symbol and from the function ψ. Thus, given a domain grammar, IDP_G, F_n can be determined for $n \in V$, the terminal symbols of the grammar, and use these values to calculate F_n for the nonterminals and SNTs of IDP_I. Given the values F_n, $E(C)$ can be calculated.

It is important to point out that, using the IDP formalism in this way enables us to analytically determine the expected cost of problem solving in situations where IDP_G and IDP_I are the same *and where they are different.* This form of analysis is valid even for grammars with very different nonterminal, SNT, and production rule sets. The only components that two grammars need to have in common is the set of terminal symbols. This is verified experimentally in Chapter 7. This is a significant result because it will enable us to conduct sensitivity analysis experiments with control architectures by adding, subtracting, or altering rules corresponding to meta-operators and then determining the expected cost of problem solving in the resulting grammar. This will provide the prediction and explanation capability necessary to develop design theories for interpretation domains.

The remainder of this thesis will demonstrate these results. In particular, the *UPC* formalism is developed which allows the characterization of control in such a way that the problem structure defined by an IDP generation grammar can be exploited by a problem solver using an evaluation function based control mechanism. After this, experimental results are shown indicating that the value of $E(C)$ determined analytically from the IDP structure of a grammar is statistically consistent with the actual results of a problem solving system. The following section will formally define the calculations of $E(C)$ and expected frequency.

5.1 Measuring the Complexity of a Domain - Calculating $E(C)$

In the IDP/*UPC* framework, the complexity of a domain can be calculated in terms of the expected cost of problem solving for a specific problem instance, $E(C)$. $E(C)$ is measured in terms of computational cost. It represents the cumulative cost of applying all operators required to generate an interpretation. This is a general measure that has several advantages. It is intuitively easy to understand compared to other measures such as expected ambiguity, which is used in the calculation of $E(C)$. $E(C)$ can be used to compare both the performance of a problem solver across different domains or different problem solvers applied to the same domain with units of measure that are consistent. Most importantly, $E(C)$ represents what is probably the most significant aspect of a problem solver's performance.

The basic approach is a three step calculation. The first step calculates the expected frequency with which states are generated corresponding to each of the elements of the grammar. (Note that the set of all state frequencies is referred to as the frequency map of the domain.) This step relies primarily on the structure of the domain as specified in the grammar and the distributions associated with the rules of the grammar. This step does not rely on the properties associated with the domain's feature list. The second stage calculates the expected probability with which paths from the states are pruned, which is called the pruning factor. This step relies both on the structure of the grammar and the domain's characteristics associated with the feature list. The final stage multiplies the expected frequency of path extensions (state frequency multiplied by pruning factor) by the expected cost of state expansion.

5.1.1 Calculating State Frequencies

The calculation of state frequencies is based on the concepts of the *singularity*, the *characteristic signal set (CSS)*, the *solution nonterminal (SNT)*, and the *sample set.* A singularity

can be thought of as a fundamental unit of analysis that repeatedly appears in a domain grammar. By first calculating the properties of a domain's singularities, it is possible to accurately and efficiently calculate the related properties of all the elements of the domain. For example, in the vehicle tracking grammars shown in this thesis, a singularity is a data point that occurs at a specific time-location. A vehicle track is not a singularity, since it spans multiple time-locations but a vehicle-location is a singularity. In the vehicle tracking grammar, other singularities include groups, signals, and noise. A more general discussion of singularities and their use in analysis is given in Appendix D.5. For a given singularity, a CSS represents the distribution of terminal symbols that can be derived from a singularity. For a vehicle-location or group singularities, the CSS would be the distribution of signal data that can be generated from it. Singularities are critical elements of the analysis analysis techniques because they are treated as independent building blocks that can be efficiently combined to determine overall domain characteristics. By determining properties of singularities, caching the values and then reusing the properties to calculate other domain properties, it is possible to calculate very accurate measure of certain domain characteristics. More specifically, if the expected values of the CVs of a singularity are known, it is possible to determine the expected values of the CVs of states that are composed of singularities. For example, in the vehicle tracking domain, if the characteristics of vehicle location singularities are known, it is possible to calculate the characteristics of tracks built using the vehicle locations. A detailed example of the use of singularities is presented in Appendix D.

An SNT is an element of the set of *solution-nonterminal* symbols that correspond to final states. For example, in a natural language domain, an SNT may contain a single element, "sentence." In the vehicle tracking domain, the SNT includes I-Track1, I-Track2, P-Track1, P-Track, G-Track1 and G-Track2.

The concept of a sample set is formally defined as follows.

Definition 5.1.1 *Sample Set, S_{sample}: A set of specific problem instances generated using IDP_G. This set can be generated in one of two ways; exhaustively or randomly. When the Sample Set is generated exhaustively, a problem instance is created for every possible combination of rules in the grammar*[1]*. For example, if nonterminal element A of a grammar has three possible RHSs for one of its production rules, a problem instance will be generated for each RHS. If the Sample Set is generated randomly, the distribution function, ψ, for IDP_G is used to create the samples in a probabilistic way. For example, if nonterminal element A of a grammar has three possible RHSs for one of its production rules, a random number is generated and this is used to determine which of the three rules to use to generate a specific problem instance.*

In addition, the following definition related to sample sets is also necessary for the calculation of state frequencies.

Definition 5.1.2 *Sample Set Weightings, w_i: A set of weighting factors associated with the elements of the Sample Set. In the case where S_{sample} is generated randomly and contains n elements, the*

[1]Note that this does *not* generate every possible string in the language defined by the grammar. Because interpretation grammars use the feature list convention, there can be many (possibly an infinite number) specific problem instances for each combination of grammar rules. Each specific problem instance is distinguished by the values instantiated for variables in its feature list. For example, though there may be a single terminal symbol that represents a specific signal data, that signal data has a location feature that could be anywhere in a sensed region. Consequently, there are many specific problem instances that are associated with the single terminal symbol.

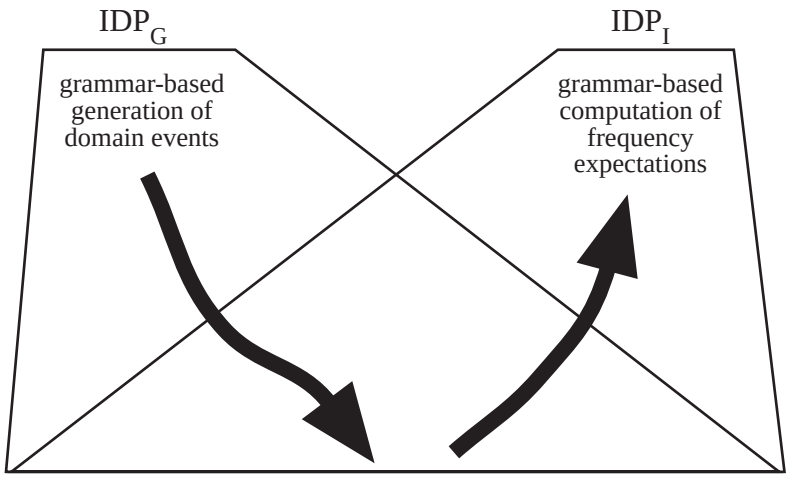


Figure 5.3. The Basic Approach to Calculating Search State Generation Frequency

Characteristic Signal Set	Distribution Factor
$\{S1, S2, S5, S7, S11, S15\}$	0.0446
$\{S2, S3, S6, S7, S11, S16\}$	0.0017
$\{S1, S3, S5, S6, S11, S16\}$	0.0017
$\{S2, S3, S4, S5, S11, S16\}$	0.0009

Figure 5.4. Example Characteristic Signal Sets for V1

weightings are $\frac{1}{n}$ for each element of S_{sample}. In the case where S_{sample} is generated exhaustively, the weighting of an element is equal to the product of the ψ values of the grammar rules used to generate the element.

The general approach to calculating state frequencies is shown in Fig. 5.3. The generational grammar, IDP_g, is used to determine the statistical distributions for CSSs. These, in turn, define the sample set and the sample set weightings. This is necessary because it is not sufficient to determine the distributions of *individual* low-level domain events. Instead, it is necessary to determine the distribution of *groups* of low-level events that can be used to generate higher-level interpretations.

Figure 5.4 shows some of the CSSs for the singularity V1 (from the vehicle tracking grammar in Fig. 11.1 from Chapter 11) and the associated distribution factors. The distributions are calculated in a top-down fashion from a grammar's ψ functions and they are used in the calculation of sample set weightings as described above. For example, the distribution factor of the first CSS in Fig. 5.4 is calculated from the distribution of the RHSs that were used to generate it. These RHSs were $V1 \rightarrow G1G3G7$ with probability = 0.4; $G1 \rightarrow S1S2$, with probability 0.45; $G3 \rightarrow S5S7$, with probability 0.45; and $G7 \rightarrow S11S15$, with probability 0.55. The distribution factor for this CSS is then $0.4 * 0.45 * 0.45 * 0.55 = 0.0446$.

The calculation of frequencies for states corresponding to nonterminals of the grammar is a recursive, or bottom-up, procedure that is based on the distribution of CSSs. Though similar, the frequency calculation differs slightly depending on the class of state. The differences in the calculation methods reflects general properties of several broad classes of states. For the vehicle tracking domain, the calculation differentiates between states that represent a single time-location (i.e., a singularity) and states that represent multiple time-locations. In addition, the calculation methodology also differentiates certain classes of meta-states. Specifically, the computations for determining the frequency of meta-states that represent clustering operations are different from those corresponding to other classes of states. The following sections formally define the calculations of state frequencies.

5.1.2 Calculating Base Frequencies for Singularities

The methods defined for calculating the frequency of singularities will be considered the standard solution approach and other methodologies will be treated as variants of this approach. This is because non-singularities should be considered "standard" grammar elements and the variations, i.e., non-singularities, clusters, etc., use frequency calculations that are derived from, and that attempt to exploit, gross properties of a grammar. This will become clear in the following sections.

Definition 5.1.3 *Base Frequency for element n of the grammar, where n is a singularity, $F^B_{n,i}$: The expected frequency of n for a given time slice of sample i, where i is an element of S_{sample}, is determined from the expected frequencies of n's children, where a "child" of a grammar element n is a terminal or nonterminal element of the grammar that appears on the RHS of one of n's production rules. The base frequency calculation is satisfactory for determining $E(C)$ in situations where there are no pruning operations. For a given production rule, p, for which n is the left-hand-side element, $F^B_{n,i}$ is calculated in a two step process. First, the expected frequency of n from each of the RHSs of p is calculated as the product of the expected frequencies of the elements of the RHSs, or $\prod_j F_{e_j}$, where each e_j is an element of the RHS. Second, the expected frequencies from each of the RHSs of p are combined by a function that is specific for n. Most combination functions are "addition." Thus, the base frequency of n from p is the sum of the base frequencies from each of the RHSs of p. In the case where n is a terminal symbol, its frequency corresponds to the number of occurrences in sample i. In the case where n is an abstract state in a projection space, the combination function often used is "maximum," which takes the maximum base frequency from the RHSs. The combination function used for these abstract states reflects the fact that in certain instances, new abstract states are not created for each combination of input data. Rather, all data is "clustered" into a single state.*

Consider the problem instance shown in Fig. 5.2 as a simple example. For the nonterminal element C, there is one production rule with two RHSs; C $\rightarrow$ fqg | fg. In this example, the frequencies of f, g, and q are all 1 and the base frequency of C associated with each RHS is $(1 * 1 * 1)$ and $(1 * 1)$ respectively. The base frequency of C derived from the rule is the sum of these frequencies, or $(1 + 1)$. As shown in the figure, there are two instantiations of C corresponding to these rules. Similarly, for nonterminal element A, there is a single rule with one RHS; A $\rightarrow$ CD. This results in a base frequency for A of $(2 * 2)$, since the frequency of D is two. As shown in the figure, there are four instantiations of A.

5.1.3 Calculating Base Frequencies for Non-Singularities

Consider the situation shown in Fig. 5.5. In this example, the grammar element t has a RHS consisting of V_t and V_{t+1}, i.e., t $\rightarrow$ V_t V_{t+1}. Given five sequential instances of V, it is possible to combine them in four distinct ways, as shown. If t is treated as a singularity, the base frequency, from Def. 5.1.3 would be computed based on $F^B_{V,i}$ * $F^B_{V,i}$. Clearly this is not correct. The actual frequency of t should be the frequency of V V combinations, given by $F^B_{V,i}$ * $F^B_{V,i}$, multiplied by the number of combinations of different V_t V_{t+1} combinations. Thus, as shown in the example, there are four combinations of V V; V_1 V_2, V_2 V_3, V_3 V_4, and V_4 V_5. Thus, the actual frequency of t is 4 * $F^B_{V,i}$ * $F^B_{V,i}$.

There are a variety of techniques available for modifying the singularity frequency calculation to work properly for non-singularity. The technique employed in this work is based on transforming non-singularities to singularities by mapping the rules of IDP$_i$ to a new grammar. (The "grammar mapping" technique was introduced in [Whitehair and Lesser, 1993] as a means for calculating a variety of measures, including potential.) The frequencies of the elements of the new grammar are then used to determine the frequency of elements of IDP$_i$.

Formally;

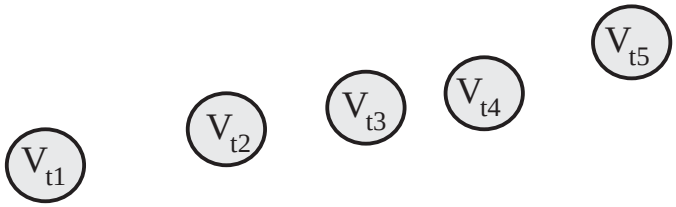
Definition 5.1.4 *Base Frequency for element n of the grammar, where n is a non-singularity, $F^B_{n,i}$: The expected frequency of n for a given time slice of sample i, where i is an element of S_{sample}, is determined by mapping rules containing n as the LHS to a new grammar, G', and then summing the base frequencies of the elements of G' corresponding to n. The relevant rules in G' are generated as follows. For each sequential time interval (t_x, t_y) that can be formed using rules with n as the LHS, create a new rule in G' of the form $n.x.y \rightarrow RHS$. Then $F^B_{n,i} = \sum_{x,y} F^B_{n.x.y,i}$, where $n.x.y$ is treated as a singularity.*

For example, again consider Fig. 5.5. Calculating $F^B_{t,i}$ would result in t being mapped to four new singularities in G', t.1.2, t.2.3, t.3.4, and t.4.5. The frequency of each of these states would be $F^B_{V,i}$ * $F^B_{V,i}$ and the frequency of t would be 4 * $F^B_{V,i}$ * $F^B_{V,i}$. If the base space grammar included the rule t $\rightarrow$ V_t V_{t+1} V_{t+2}. t would be mapped to the singularities t.1.3, t.2.4, and t.3.5 in G'. The frequency of each of these would be $F^B_{V,i}$ * $F^B_{V,i}$ * $F^B_{V,i}$.

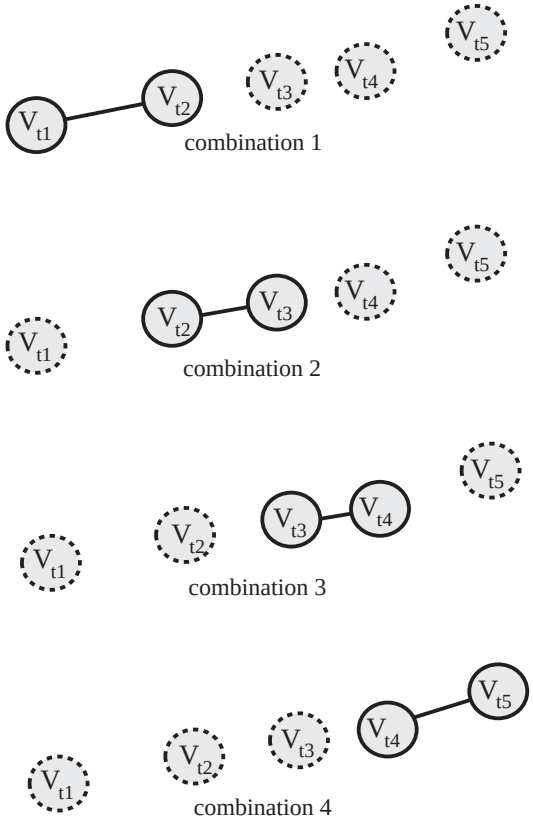
5.1.4 Adjusting Base Frequencies for Pruning

When pruning operations are available, the base frequency calculation must be modified appropriately. The following definitions are used to accomplish this by factoring in the probability that a state will be pruned based on its credibility. If other pruning operations are available, such as an operation that would prune a state based on its probability of being included in a final solution, they would be factored into the calculation of frequency in a similar way.

Definition 5.1.5 *Expected credibility of element n of the grammar, $\mu_{Cred}(n) = \sum_i \mu_{f_{p.i}} * \psi(p.i)$, where $\mu_{f_{p.i}}$ is the expected credibility of the evaluation function $f_{p.i}$ and $\psi(p.i)$ is the distribution of the rules $p.i$. In the experiments described in this thesis, $\mu_{f_{p.i}}$ is calculated by assuming a distribution for the credibility functions $\Gamma_{p.i}$ that is normal with mean equal to the average of the means of the inputs. This enables us to recursively calculate the expected credibility of each element of the grammar based on the expected credibilities of an element's children.*



a. initial vehicle level data from 5 time periods



b. all possible partial tracks sequences spanning two consecutive time periods generated by the rule t -> VV

Figure 5.5. Calculating the Frequency of Non-Singularities

Definition 5.1.6 *Expected standard deviation from $\mu_{Cred}(n)$ for element n of the grammar, $\sigma_{Cred}(n) = ftn(\sigma_{f_{p.i}})$, where ftn is a function of the variances of the credibility rules, $f_{p.i}$ and is based on the standard equation $Var(X+Y) = VarX + VarY + 2 * Cov(X,Y)$. In the experiments described in this thesis, it is assumed that variances for the credibility functions Γ_p that is equal to sum of the variances of the inputs. (The standard deviation is then the square root of the variance.) It is assumed that the credibilities and variances of siblings are independent, and this simplifies to $Var(X+Y) = VarX + VarY$. As with expected credibility, $\sigma_{Cred}(n)$ can be calculated recursively for each element of the grammar.*

Definition 5.1.7 *Pruning Modifier for element n of the grammar, $Pf_n = (1 - P(Credibility(n) \leq T))$, where $Credibility(n)$ is the credibility of the search state corresponding to n that is determined dynamically at run time, and T is the pruning threshold[2]. Given that normally distributed credibilities have been assumed, this value can be calculated from $\phi(\frac{T-\mu_{Cred}(n)}{\sigma_{Cred}(n)})$, where ϕ is available in standard probability textbooks and is equal to $\frac{1}{\sqrt{2\pi}} * e^{-x^2/2}$.*

Definition 5.1.8 *Expected frequency for element n of the grammar, $F_n = \sum_i w_i * (F_n^B * (1 - Pf_n))$, where i is an element of S_{sample}, w_i the weighting of i in S_{sample}, and $F_{n,i}^B$ the base frequency of n. The expected frequency of n is determined by determining the base frequency of n for element of the Sample Set, modifying this value to reflect pruning actions, multiplying by the sample weight to normalize the value, then summing all values.*

5.1.5 Calculating Precedence Relations

The calculation of $E(C)$, as well as other calculations, depends on the concept of *precedence* relations. Precedence relations implicitly define the relative order in which operators are applied and they can be used to determine the state of problem solving at a given time, t.

For example, consider the grammar shown in Fig. 5.6. This grammar is also shown in Fig. 5.7, which illustrates precedence relations more clearly. In grammar G, state A will always be generated after states C and D, in situations where A can be generated. This is represented this with the precedence relation, $C < A$ and $D < A$. Furthermore, states C and D will always be generated after states f, g, h and i.

The table in Fig. 5.8 shows some of the precedence relations for grammar G. The symbols across the left side and the top of the table correspond to symbols of the grammar. The table cell indicates the relationship between the grammar symbols. For example, from the second row of the table, $C < A$, $D < A$, etc. The table is constructed by first specifying the direct relations between grammar elements that appear on the LHS of a rule and the elements that appear on the RHS, then taking the transitive closure of these relations.

Notice the relation $z <_{0.5} W$. The subscript indicates that this is not a *hard* precedence relation. Rather, a W can be generated without a z, but the z will be generated before the W 33% of the time. The generation of a W will precede the generation of a z in situations where operator $op_{6.1}$ is applied before operator op_{15}. In this particular system, it is assumed that the operators are all given equal ratings, so this will occur randomly 1 time out of 3.

---

[2] In later chapters we introduce bounding functions with thresholds that are determined dynamically. In these situations, pruning modifiers are computed based on T equal to the expected credibility of a "correct" result for a given problem instance.

Interpretation Grammar *G'*

grammar rule	distribution	credibility	cost
0.1 $S \rightarrow A$	$\psi(0.1) = 0.2$	$f_{0.1}(f_A)$	$g_{0.1}(g_A)$
0.2 $S \rightarrow B$	$\psi(0.2) = 0.2$	$f_{0.2}(f_B)$	$g_{0.2}(g_B)$
0.3 $S \rightarrow M$	$\psi(0.3) = 0.2$	$f_{0.3}(f_M)$	$g_{0.3}(g_M)$
0.4 $S \rightarrow N$	$\psi(0.4) = 0.2$	$f_{0.4}(f_N)$	$g_{0.4}(g_N)$
0.5 $S \rightarrow O$	$\psi(0.5) = 0.2$	$f_{0.5}(f_O)$	$g_{0.5}(g_O)$
1. $A \rightarrow CD$	$\psi(1) = 1$	$f_1(f_C,f_D, \Gamma_1(C,D))$	$g_1(g_C,g_D,C(\Gamma_1(C,D)))$
2. $B \rightarrow DEW$	$\psi(2) = 1$	$f_2(f_D,f_E,f_W, \Gamma_2(D,E,W))$	$g_2(g_D,g_E,g_W,C(\Gamma_2(D,E,W)))$
3.0 $C \rightarrow fg$	$\psi(3.0) = 0.5$	$f_{3.0}(f_f, f_g, \Gamma_{3.0}(f,g))$	$g_{3.0}(g_f, g_g,C(\Gamma_{3.0}(f,g)))$
3.1. $C \rightarrow fgq$	$\psi(3.1) = 0.5$	$f_{3.1}(f_f,f_g,f_q, \Gamma_{3.1}(f,g,q))$	$g_{3.1}(g_f,g_g,g_q,C(\Gamma_{3.1}(f,g,q)))$
4. $E \rightarrow jk$	$\psi(4) = 1$	$f_4(f_j,f_k, \Gamma_4(j,k))$	$g_4(g_j,g_k,C(\Gamma_4(j,k)))$
5.0 $D \rightarrow hi$	$\psi(5.0) = 0.5$	$f_{5.0}(f_h,f_i, \Gamma_{5.0}(h,i))$	$g_{5.0}(g_h,g_i,C(\Gamma_{5.0}(h,i)))$
5.1. $D \rightarrow rhi$	$\psi(5.1) = 0.5$	$f_{5.1}(f_r,f_h,f_i, \Gamma_{5.1}(r,h,i))$	$g_{5.1}(g_r,g_h,g_i,C(\Gamma_{5.1}(r,h,i)))$
6.0 $W \rightarrow xyz$	$\psi(6.0) = 0.5$	$f_{6.0}(f_x,f_y,f_z, \Gamma_{6.0}(x,y,z))$	$g_{6.0}(g_x,g_y,g_z,C(\Gamma_{6.0}(x,y,z)))$
6.1. $W \rightarrow xy$	$\psi(6.1) = 0.5$	$f_{6.1}(f_x,f_y, \Gamma_{6.1}(x,y))$	$g_{6.1}(g_x,g_y,C(\Gamma_{6.1}(x,y)))$
7. $f \rightarrow (s)$	$\psi(7) = 1$	$f_7(f_{(s)}, \Gamma_7((s)))$	$g_7(g_{(s)},C(\Gamma_7((s))))$
8. $j \rightarrow (s)$	$\psi(8) = 1$	$f_8(f_{(s)}, \Gamma_8((s)))$	$g_8(g_{(s)},C(\Gamma_8((s))))$
9. $g \rightarrow (s)$	$\psi(9) = 1$	$f_9(f_{(s)}, \Gamma_9((s)))$	$g_9(g_{(s)},C(\Gamma_9((s))))$
10. $k \rightarrow (s)$	$\psi(10) = 1$	$f_{10}(f_{(s)}, \Gamma_{10}((s)))$	$g_{10}(g_{(s)},C(\Gamma_{10}((s))))$
11. $h \rightarrow (s)$	$\psi(11) = 1$	$f_{11}(f_{(s)}, \Gamma_{11}((s)))$	$g_{11}(g_{(s)},C(\Gamma_{11}((s))))$
12. $x \rightarrow (s)$	$\psi(12) = 1$	$f_{12}(f_{(s)}, \Gamma_{12}((s)))$	$g_{12}(g_{(s)},C(\Gamma_{12}((s))))$
13. $i \rightarrow (s)$	$\psi(13) = 1$	$f_{13}(f_{(s)}, \Gamma_{13}((s)))$	$g_{13}(g_{(s)},C(\Gamma_{13}((s))))$
14. $y \rightarrow (s)$	$\psi(14) = 1$	$f_{14}(f_{(s)}, \Gamma_{14}((s)))$	$g_{14}(g_{(s)},C(\Gamma_{14}((s))))$
15. $z \rightarrow (s)$	$\psi(15) = 1$	$f_{15}(f_{(s)}, \Gamma_{15}((s)))$	$g_{15}(g_{(s)},C(\Gamma_{15}((s))))$
16. $M \rightarrow Y$	$\psi(16) = 1$	$f_{16}(f_Y)$	$g_{16}(g_Y)$
17.0 $Y \rightarrow qr$	$\psi(17.0) = 0.5$	$f_{17.0}(f_q,f_r, \Gamma_{17.0}(q,r))$	$g_{17.0}(g_q,g_r,C(\Gamma_{17.0}(q,r)))$
17.1 $Y \rightarrow qhri$	$\psi(17.1) = 0.5$	$f_{17.1}(f_q,f_h,f_r,f_i, \Gamma_{17.1}(q,h,r,i))$	$g_{17.1}(g_q,g_h,g_r,g_i,C(\Gamma_{17.1}(q,h,r,i)))$
18. $N \rightarrow Z$	$\psi(18) = 1$	$f_{18}(f_Z)$	$g_{18}(g_Z)$
19. $Z \rightarrow xy$	$\psi(19) = 1$	$f_{19}(f_x,f_y, \Gamma_{19}(x,y))$	$g_{19}(g_x,g_y,C(\Gamma_{19}(x,y)))$
20. $O \rightarrow X$	$\psi(20) = 1$	$f_{20}(f_X)$	$g_{20}(g_X)$
21.0. $X \rightarrow fgh$	$\psi(21.0) = 0.5$	$f_{21.0}(f_f,f_g,f_h, \Gamma_{21.0}(f,g,h))$	$g_{21.0}(g_f,g_g,g_h,C(\Gamma_{21.0}(f,g,h)))$
21.1. $X \rightarrow fg$	$\psi(21.1) = 0.5$	$f_{21.1}(f_f,f_g, \Gamma_{21.1}(f,g))$	$g_{21.1}(g_f,g_g,C(\Gamma_{21.1}(f,g)))$

(s) = signal data $\Gamma_n(i,j,...)$ = semantic evaluation function for rule n $C(\Gamma_n(i,j,...))$ = cost of executing $\Gamma_n(i,j,...)$

Figure 5.6. Interpretation Grammar *G* with Fully Specified Distribution, Credibility, and Cost Functions

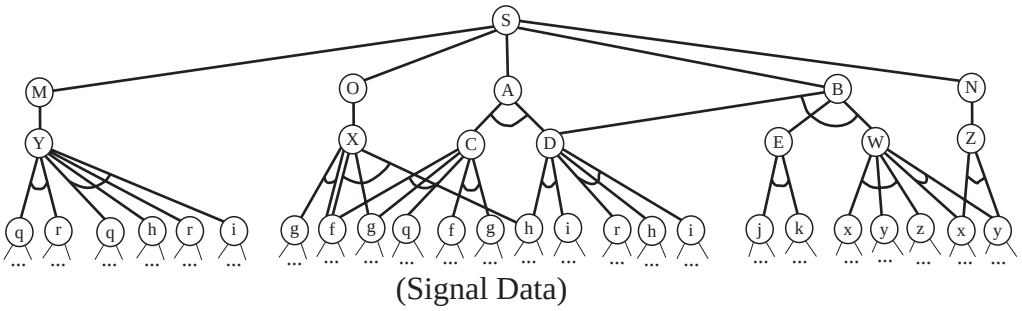


Figure 5.7. Interpretation Grammar *G*

	S	A	B	C	D	E	W	f	g	j	k	h	i	x	y	z
S																
A				<	<			<	<			<	<			
B					<	<	<			<	<	<	<	<	<	<0.33
C								<	<							
D												<	<			
E										<	<					
W														<	<	<0.33
f																
g																
j																
k																
h																
i																
x																
y																
z																

Figure 5.8. Precedence Relations for Grammar *G*

Precedence relations are used to calculate the probability that one state is generated before another. This is used in the calculation of pruning factors. In addition, precedence relations can be used to determine the *expected state of problem solving* for a given time, t. Problem solving state can be determined using the expected cost of applying each operator and an expected ordering of the operators determined implicitly from the precedence relations.

For example, assume that each of the operators in G has a cost of 1. Then the expected state of problem solving can be determined statistically using the precedence relations. For example, the probability of the problem solver generating an S before time 5 is 0. This is because the shortest path to S is by generating an M from a Y which is generated from a q and an r. Since the cost of generating the q, r, Y, and M is each 1, and the cost of generating the S is also 1, it is not possible to generate the S before time 5. By statistically analyzing the distribution of domain events, it is possible to characterize the state of problem solving at any given time, t.

5.2 Calculating Expected Operator Cost

In the experiments described in later chapters, the cost of an operator is modeled as a fixed number. This is not a restriction on the analysis framework. These numbers are used in the experiments presented here so that it is possible to verify the results using hand calculations. In general, the cost of an operator will include a variable portion, specified as a mean and a variance, and a fixed portion. For example, Cost(op_i) = $3*$(number of inputs)$+10$. The variable and fixed cost values are components of an IDP_i grammar specification.

5.3 Calculating Expected Correct Answers

The explicit representation of pruning functions supports the calculation of the expected number of correct answers that are pruned in situations when they should not be. This is simply the sum of the probabilities that paths will be pruned when they are components of "correct" interpretations.

In the interpretation domains that are studied in this thesis, there are two different perspectives on what a correct answer is. The definition used in the IDP/*UPC* framework is that the correct answer is the interpretation with the highest credibility. An alternative perspective is that the correct answer is the interpretation that corresponds to the events that generated the signal data. The differences may appear subtle but are actually quite significant. This is because the interpretation with the highest rating is not guaranteed to correspond to the events that generated the signal data.

5.4 Calculating $E(C)$ and Expected Frequencies

Given the expected frequency, it is possible to calculate $E(C)$ by multiplying the expected cost associated with each search state by the frequency of the state and summing for all states.

Definition 5.4.1 *Expected cost associated with search state n, $E_{cost}(n) = \sum_p \bar{g}_p$, where p is a production rule, n is an element of some RHS of p, and $\bar{g}_p$ is the expected cost of applying the search operator corresponding to p. To determine the expected cost associated with a state, the expected costs of all the operators that can be applied to the state are summed. An operator can be applied to a state if the grammar element associated with the state appears in any of the RHSs of the grammar*

production rule associated with the operator. Since a state is connected only when all paths that lead from it are either extended to final states, extended to dead end states, or pruned, this sum represents the cost of connecting a state.

Definition 5.4.2 *Expected cost of problem solving,* $E(C) = \sum_n F_n * E_{cost}(n)$. *The expected cost of problem solving is determined by summing, for each element of the grammar, the expected frequencies of the grammar elements multiplied by the expected cost associated with connecting the corresponding search state.*

The procedure used to calculate $E(C)$ can be summarized as:

1. Generate the Sample Set, either randomly or exhaustively.
2. Generate the sample weightings.
3. Using a bottom-up recursion, calculate, in order, the following for each element of the grammar:
   - base frequency of the element
   - expected credibility of the element
   - variance from expected credibility of the element
   - pruning modifier of the element
   - expected frequency of the element
4. Calculate the expected costs associated with connecting search states corresponding to grammar elements.
5. Calculate $E(C)$.

5.5 Chapter Summary

This chapter defines quantitative analysis tools based on statistical properties of IDP problem and problem solver specifications. These tools can be used to calculate characteristics such as the expected cost of a problem solving instance, the expected frequency with which partial and full interpretations will be generated, the expected utilities of partial and full interpretations, the relative ordering of problem solving actions, and the expected number of correct answers that are eliminated by pruning operators. These measures are significant from an analytical perspective because they measure important properties of a problem solver's performance and they are significant from a conceptual perspective because they demonstrate that relevant quantitative analysis can be conducted using the IDP formalism.

CHAPTER 6

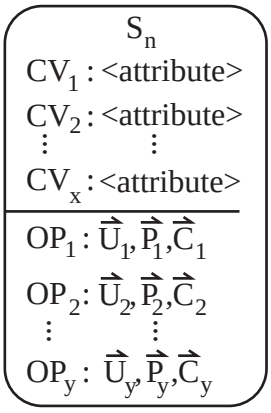
IMPLEMENTING IDP BASED PROBLEM SOLVING SYSTEMS – THE *UPC* MODEL

Chapter 3 presented the IDP formalism and Chapters 4 and 4.8 demonstrated how it can represent characteristics of a domain's problem structure. We will now begin to develop the analytical tools that will be used to explain and predict the effectiveness of specific control architectures that use abstractions. The essence of these analyses is that control architectures that use abstractions exploit problem structures and these structures, along with the abstractions used by the control component, can be represented naturally in the IDP formalism. In this section, another formalism, the *UPC* model, is introduced to extend the analytical power of IDP models. In the analysis IDP/*UPC* framework, the *UPC* and IDP formalisms are closely linked – the IDP formalism models the structure of a domain theory and the *UPC* formalism maps IDP structures to a search space model where the structures are explicitly represented in a manner that can be used in a control architecture's evaluation function. This mapping is based on the local perspective of problem solving of a specific state in the search space and on statistical properties of the domain derived from the formal IDP description. It does not take into account the existence or absence of any other states. Using the *UPC* representation, we can construct problem solving systems capable of achieving the levels of performance predicted by quantitative analysis of IDP domain specifications and the optimal interpretation control strategy. As a consequence, domains with different structures can be compared using identical evaluation function based control architectures or these architectures can be varied to compare performance of different problem solvers within a given domain.

Though the IDP and *UPC* models are closely linked in the analysis framework presented in this thesis, they can both be used as stand alone analysis tools. Consequently, the *UPC* formalism will first be presented as a general analysis tool, and subsequent chapters will discuss the integration of the IDP and *UPC* models into a powerful framework for the analysis of sophisticated control. The initial presentation of the *UPC* as an independent analysis tool will include examples from IDP models and interpretation problems. This is intended to provide a frame of reference and simplify later discussion. It is not intended to imply that the use of the *UPC* model is restricted to interpretation tasks or even that its use be in conjunction with the IDP model.

6.1 Overview of the *UPC* Formalism

The *UPC* model is an extension to the traditional search paradigm that explicitly represents important characteristics of search spaces that are often used implicitly in control architectures. *UPC* models explicitly represent where in a search space a given state lies relative to potential final states, the uncertainty associated with the ability to reach each final state, the expected cost



Extended State representation: operators applicable to each state are represented with the corresponding *Utility, Probability,* and *Cost* vectors.

Each vector entry consists of a measure of the *expected value* and a measure of the *variance.*

Expected values and variances are determined from IDP's cost and credibility functions. (In interpretation problems, credibility = utility.)

Figure 6.1. Representation of a Search State in the *UPC* Model

of reaching the final state, and the final state's expected value, or *utility*[1]. The *UPC* information forms the basis for evaluation functions. In interpretation problems, this information can be derived from the specification of a domain theory's structure from Chapters 3 and 4, in particular, the functions defining a domain theory's credibility and cost structures.

In a *UPC* model, a search space is defined by *characteristic variables*, or *CVs*, that specify the properties of individual states, operators that map (generate) one state to (from) another, and functions of CVs that define final states. The specification of operators is identical to the traditional notion of operators. The specification of the start state, intermediate states, and final states is similar except for an extension to their representations. The extension expands the set of CVs that characterize a state to include a set of vectors, $\{U, P, C\}$, that are characteristics of the operators that can be used to extend a state. Figure 6.1 is a representation of a search state based on the *UPC* model.

Intuitively, U, P, and C can be thought of as the characteristics that determine the desirability of expanding a given state. They represent the cost and utility structure of a space in a way that can be exploited by the control component to determine which state is the best to expand and which operator to use. The U, P, and C vectors associated with each operator can be thought of as the inputs to the evaluation function that orders problem solving activity. In addition, U, P, and C will be used to formalize control architectures that exploit problem structures such as *uncertainty, redundancy, interacting subproblems*, and *semi-monotonicity and bounding functions* (as defined in Chapter 4.8) within the search paradigm.

U, P, and C provide a map or coordinate system defining each state's location in the search space structure defined by the cost and credibility functions associated with the IDP's characteristic grammar, G. U specifies the direction of a search path (in terms of final states, or interpretations, they lead to) and P defines the probability that the path can be traversed successfully. Thus, every potential solution path[2] containing state s_n is represented by an entry in s_n's U vector that has a corresponding nonzero entry in s_n's P vector. For each path defined by s_n's U and P vectors, C defines *where* on the path s_n lies. If the C entry is small, then s_n is close to the final state. If the C entry is large, then s_n is far from the final state.

[1] In the IDP formalism, utility can be thought of as a state's credibility.

[2] A solution path is a derivation path that leads to an interpretation.

6.2 *UPC* States and the IDP Formalism

Using the *UPC* formalism, search states are created dynamically as problem solving operators are applied to existing states. The *UPC* vectors are determined dynamically, when a state is created, and included in the state's CV specification as previously described. In addition, each state corresponds to an element from the IDP grammar. This correspondence is not necessarily one-to-one. For each element of the IDP grammar, there may be many different corresponding *UPC* states. Each of the states is differentiated by its distinct set of CVs which might include information such as "supporting data," "time of occurrence," "location," etc.

It is important to stress that the *UPC* representation accounts for multiple occurrences of a specific interpretation. In an actual run, multiple instances of an interpretation can be constructed resulting in distinct paths that must be accounted for when computing expected cost. Multiple instances of an interpretation result from ambiguities in a domain grammar that allow a single set of data to be interpreted in a variety of different ways. This is discussed at length in previous chapters.

6.3 *UPC* Representation

The *UPC* model specifies that the following vectors are defined for every state in the search space. As shown in Fig. 6.1, the vector definitions for a given operator are a component of the corresponding state's CV set. Consequently, the *UPC* vectors are incorporated in definitions of the search space's structure.

U_{s_n} = set of *utility expectation* vectors. For each operator op_i that can be applied to state s_n, $U_{s_n}(op_i)$ defines a vector where each element is a value defining the *expected utility*[3] of the j^{th} final state that can be reached from s_n via a path beginning with operator op_i. This vector will also be represented as $u_{n,i}$. In terms of the IDP formalism, elements of the vectors in U_{s_n} correspond to high-level interpretations, T, that correspond to SNTs from the grammar that include the subtree s_n and the utility expectation can be thought of as T's expected credibility or a function of T's expected credibility. U_{s_n} is exhaustive in the sense that *all* final states, as defined by the SNTs of the grammar, that can be reached via a path including the states that are created by applying op_i to s_n are represented by an element in some $u_{n,i}(j) \in U_{s_n}$. In general, the use of SNTs in this role is artificial. The *UPC* values can be thought of as coordinates of a state's position in a search space relative to potential final states. The results that are derived here can also be derived without the concept of SNTs by using only the start symbol in the calculations. However, SNTs have been developed and are used in calculations because they provide a more intuitive basis for understanding the values that are computed. The use of SNTs clarifies the representation somewhat for explanation purposes, and it greatly simplifies many system implementation issues (primarily debugging issues).

As defined in Chapter 3.2, the utility structure of a domain is defined recursively in terms of the production rules of a grammar. The interpretation problems discussed in this thesis use the convention that a state's utility is a function of the utilities of its

[3] It is important to note that the corresponding variance of the expected utility is a meaningful consideration within this formalism. However, for the sake of clarity, its significance will be discussed in a separate context. The variance associated with the other vectors, P and C will be similarly treated.

immediate descendants and the semantic function Γ. For this thesis, a state's credibility is computed to be the average of these values. Thus, the expected utility of a given state *B*, and its variance, is derived from the expected utilities and variances generated by *B*'s RHS production rules, p_i, and by the expected value and variance of the function Γ_p. As described in Chapter 3.2, the utility (credibility) of a state generated by grammar rule i is computed by the function f_i.

More formally,

Definition 6.3.1 *Expected utility of a state* = $\sum_{i=1}^{n} \psi(i) * \mu_i$, *where* n = *number of RHS productions for the state,* $\psi(i)$ = *the frequency associated with production* i, *and* μ_i = *expected value of the utility function,* f_i, *associated with production* i.

These values are used to compute the expected utilities of the elements in U_{s_n}. For a given state, s_n, and operator, op_i, the expected utilities of the final states that can be reached along paths from s_n beginning with op_i are computed as discussed in Chapter 3.2 by using the actual utility (or credibility) of s_n and the expected utilities of other relevant states.

P_{s_n} = set of *conditional probability* vectors. $p_{n,i}(j)$ represents the likelihood that a path beginning with op_i can be constructed from s_n to the final state corresponding to the expected utility $u_{n,i}(j)$. Using the IDP model, elements of vectors in P_{s_n} can be determined from the distribution function ψ for the IDP's characteristic grammar, G, and from the expected distribution of domain events.

It is important to note the difference between U_{s_n} and P_{s_n}. The entries in the vectors P_{s_n} in no way indicate the 'worth' or 'utility' of a particular state or path. They only indicate the probability of reaching a specific final state via a path that includes s_n and begins with op_i. Thus, a path with a high-probability is not necessarily correct and a path with a low-probability is not necessarily incorrect.

Similarly, the entries in the vectors U_{s_n} do not specify the likelihood that a particular final state can be reached other than to indicate that the probability is nonzero, in which case the final state's utility is represented in a vector in U_{s_n}. Rather, the entries only represent the expected worth (or credibility) of a particular final state. Thus, a path with high-credibility does not necessarily have a high-probability and a path with low-probability does not necessarily have a low-probability.

The definitions of $u_{n,i}$ and $p_{n,i}$ lead to the following (Note: in these definitions, upper case letters indicate elements of a grammar's set of SNTs, which correspond to the final states in that associated search space.):

Definition 6.3.2 *The expected utility of a path from state n to final state j that begins with* $op_i = u_{n,i}(j) * p_{n,i}(j)$. *i.e., The expected utility of a final state multiplied by the conditional probability of successfully reaching the final state given state* n.

Definition 6.3.3 $p_{n,i}(j) = P(j \mid n) R_{prune}(n, j)$. *The likelihood that a path exists from state n to final state j that begins with* op_i *is equal to the conditional probability that final state* j *can be generated given an* n *multiplied by a* pruning factor, $R_{prune}(n, j)$, *which*

represents the probability that the path is not pruned by a bounding function before final state j is generated.

The pruning factor, $R_{prune}(n, j)$, is only used in domains that include bounding operators that are based on dynamic pruning thresholds. In such domains, $R_{prune}(n, j)$ must be computed dynamically. In domains that do not include bounding operators, the pruning factor is 1. The computation of $R_{prune}(n, j)$ is based on a domain statistic, Ω, that represents the probability that a path to a state is pruned by a bounding function in a situation where the path can be created. Ω is defined for elements of the grammar that correspond to bounding functions, to nonterminals that do not correspond to bounding functions, and to terminals. All definitions of Ω are based on the a priori computations for the expected values of the credibilities of the elements of a grammar. For nonterminal element s that does not correspond to a bounding function, the definition of Ω is:

Definition 6.3.4 $\Omega(s) = \Sigma_i \psi(i) * \Pi_j \Omega(n_j)$. *Where each i corresponds to a production rule with s as a left-hand-side and each n_j is an element of the right-hand-side of the production rule.*

For nonterminal element s that corresponds to a bounding function, the definition of Ω is:

Definition 6.3.5 $\Omega(s) = (1 - P(credibility(n) < t) * \Omega(n)$. *Where n is the right-hand-side of the production rule, t is the pruning threshold for the bounding function, and $credibility(n)$ is the expected credibility of grammar element n.*

For terminal element s, $\Omega(s) = 1$.

The computation of $R_{prune}(n, j)$ is based on the credibility of n and the values of Ω that are computed for the grammar. Formally,

Definition 6.3.6 $R_{prune}(n, j) = \Omega(j, n)$. *Where $\Omega(j, n)$ in the computation of $\Omega(j)$ with respect to n.*

The computation of $\Omega(j, n)$ is straightforward, but it must be done dynamically using the actual credibility of n instead of the a priori expected credibility. This is defined as:

Definition 6.3.7 $\Omega(s) = \frac{\Sigma_i(\psi(i) * \Pi_j \Omega(x_j, j))}{\Sigma_i \psi(i)}$. *Where each i corresponds to a production rule with s as a left-hand-side that includes n in a derivation tree, each x_j is an element of the right-hand-side of the production rule. The sum, $\Sigma_i \psi(i)$ is used to normalize the computation relative to n. When computing $\Omega(x_j, j)$, if n is not included in any of the interpretation trees for x_j, the value $\Omega(x_j)$ is used.*

In general, conditional probabilities can be computed by:

Definition 6.3.8 $P(A \mid b) = \frac{P(A \cap b)}{P(b)}$. *Conditional probability of state A given state b, where b is a descendant of A (i.e., b is on the RHS of some set of grammar rule applications that begin with A on the LHS). This is the conventional definition of conditional probability that is available in any appropriate textbook.*

For IDP models, the following equations can be used to determine conditional probabilities:

Definition 6.3.9 $P(A) = P(S \rightarrow A)$[4], *domain specific distribution functions. Probability of the domain event corresponding to interpretation A occurring. This probability will be specified with domain specific distribution functions. In general, these distributions will be represented with production rules of the grammar associated with the start symbol. The RHSs of these rules will be from the grammar's set of SNTs. Uncertainty regarding this distribution leads to problem solving uncertainty.*

Definition 6.3.10 $\{RHS(A)\}$ = *the set of elements that appear on right-hand-sides of production rules with A on the left-hand-side.*

Definition 6.3.11 $P(b \in \{RHS(A)\}^{\vdash +}) = \sum_{\forall i} P(b \in RHS_i(A)) + \sum_{\forall r'}(P(r' \in \{RHS(A)\}) * P(b \in \{RHS(r')\}^{\vdash +}))$, *where* $\{RHS(A)\}$ *is the set of all RHSs of A,* $RHS_i(A)$ *is the RHS of the* i^{th} *production rule of A,* $P(b \in RHS_i(A)) = \psi(RHS_i(A))$ *if* $b \in RHS_i(A)$, *0 otherwise, and each element* r' *is a nonterminal that appears in a RHS of A that does not also include* b. *The probability of partial interpretation b being included in any RHS of A, as defined by the distribution function* $\psi(A)$. *The "*$\vdash +$*" notation indicates that the definition of RHS is recursive. i.e.,* $\{RHS(A)\}^{\vdash +}$ *represents the transitive closure of all states that can be generated from A. Thus, b can be in an RHS of A, or in the RHS of some element of an RHS of A, etc.*

Definition 6.3.12 $P(b) = P(S \rightarrow b) = \sum_{\forall A} P(A) * P(b \in \{RHS(A)\}^*)$. *Probability of partial interpretation b being included in an interpretation.*

Definition 6.3.13 $P(A \rightarrow b) = P(A) * P(b \in \{RHS(A)\}^*)$. *Probability that the partial interpretation, b, is generated from full or partial interpretation A, where b is a descendant of A.*

Definition 6.3.14 *Ambiguity – Given a domain event, A, its interpretation is* ambiguous with *the interpretation of a second domain event, B, when B* subsumes *A (the subsume relationship is specified in Definition 4.4.5 in Chapter 4). i.e., A is ambiguous with B when* $B \Rightarrow A$. *(The low-level signal data generated by B can be mistaken for an A.) Note that this definition of ambiguity is* not *reflexive. Thus, A being ambiguous with B does not imply that B is ambiguous with A. This definition of ambiguity is consistent with Definition 4.1.1 and will be used where appropriate.*

[4]The notation used in these equations, i.e., $S \rightarrow A$ is distinct from the production rule notation used to designate grammar rules and should not be confused as production rule notation.

Definition 6.3.15 $P(A \cap b) = P(A \rightarrow b) + \sum_{\forall B} P(B \rightarrow b)$. *Intersection of domain events A and b, where b is a descendant of A, and where the interpretation of A is* ambiguous with *the interpretation of each B. The intersection of A and b will occur when both A and b are generated during the course of a specific problem solving instance. This will occur when A leads to the generation of b and when the occurrence of a distinct event, B, leads to the generation of b and when A is ambiguous with B. In the case where B leads to the generation of b, b and A still intersect because an A will be generated during processing since A is ambiguous with B.*

C_{s_n} = set of *cost* vectors. $c_{n,i}(j)$[5] is a pair of values, represented $c_{n,i}(j,1)$ and $c_{n,i}(j,2)$. The first is the expected cost of generating the path, *when the path can be generated*, from s_n, beginning with op_i, to the final state corresponding to $u_{n,i}(j)$. The second is the expected cost of extending the path *when the final state cannot be reached.* These two values can be used to specify the expected cost of *attempting* to complete a path. Formally,

Definition 6.3.16 *For state s_n, the expected cost of the path corresponding to $u_{n,i}(j)$ is* $p_{n,i}(j) * c_{n,i}(j,1) + (1 - p_{n,i}(j)) * c_{n,i}(j,2)$.

Definition 6.3.17 *The expected cost of a "correct" path* = $c_{n,i}(j,1) = E(cost(F)) - E(cost(s_n))$, *where F = $u_{n,i}(j)$ – the final state the path is attempting to reach and $E(cost(s_n))$ represents the expected cost of generating state s_n. (Using the notation from Chapter 6.4, cost functions would be represented $g(u_{n,i}(j)) - g(s_n)$.) Thus, the estimated cost of reaching the final state (completing the interpretation tree, T) is a function of the cost of deriving the entire interpretation tree minus the cost already incurred to derive the subtree corresponding to s_n.*

Definition 6.3.18 *The expected cost of a specific "incorrect" path =* $C_{\neg F,R}(s_n) = \sum_{\forall m} P((n \cap m) \mid R) * (E(cost(s_m)) - (\sum_{\forall t} E(cost(s_t)) * P(s_m \rightarrow s_t)))$, *where $E(cost(s_m))$ is the expected cost of generating state s_m, the set m comprises the components of F that are not also components of s_n, and the set t comprises the components of each s_m. The term $E(cost(s_m)) - (\sum_{\forall t} E(cost(s_t)) * P(s_m \rightarrow s_t))$ is the incremental cost of each of the components of F. Thus, this definition takes into consideration the probability of each component of F being generated and the incremental cost of each component. Intuitively, $C_{\neg F,R}$ represents the expected cost the problem solver will incur before determining that a path cannot be generated to F. It is necessary to differentiate each R, since the cost of a failed path to F will vary with the different R. For example, in some situations, the cost of a failed path to F may be very small. This may occur when interpretations of "B" are correct. In contrast, the cost of a failed path to F may be very large when the correct interpretation is "C." This will become clear in Chapter 6.5.*

Definition 6.3.19 *The expected cost of connecting all "incorrect" paths from a state =* $c_{n,i}(j,2) = \sum_{\forall R} \frac{P(R \rightarrow n)}{(P(n) - P(F \cap n))} * C_{\neg F,R}(n)$, *where F represents the final state corresponding*

[5] The subscript n is used to represent s_n in order to simplify the notation.

to $u_{n,i}(j)$, $C_{\neg A,R}(n)$ *is the* expected cost *of a failed attempt to generate a path to* F *given that* R *is the correct interpretation,* R *is an element of the set SNT, and where* F *is not ambiguous with* R.

In the case where the path does not exist, the expected cost will be dependent on the structure of the domain. In domains with a great deal of ambiguity, this value could be almost as large as (or perhaps much larger than) the expected cost of a correct path. Examples of computing expected costs for paths are presented in Chapter 6.5.

6.4 A Basis for Analysis - An *Optimal Objective Strategy*

As discussed in Chapter 1.7, the analysis paradigms supported by the IDP/*UPC* framework all involve the use of four elements: a problem's structure and a problem solver's objective strategy, control architecture, and performance level (or behavior). Chapters 3 and 4 presented the IDP formalism which uses a unified representation to describe a problem's structure and the abstractions and approximations used by a problem solver's control architecture. Furthermore, as was discussed in Chapter 1.7 and as will be discussed in the Chapter 7, the performance of a problem solver will be measured in terms of the expected cost of problem solving and the expected probability that the problem solver will find the correct answer, where the correct answer is defined to be the highest rated interpretation. This section defines the fourth element, the objective strategy, that is needed in the analysis paradigms. Chapter 7 shows how this control architecture can be used with an expanded state representation that explicitly represents certain quantitative properties of a search space that are derived from an IDP specification. We will then experimentally verify the quantitative results determined analytically with the IDP formalism in Chapter 5.

6.4.1 Defining Optimal Interpretations

There are a variety of broad objective strategies that could be used as a basis for the analysis paradigms. For example, one class of objective strategies is related to finding *any* solution as quickly as possible, another class of strategies is related to finding the least cost solution. These, and other general objective strategies are discussed more formally in Appendix A.

The strategy that will be defined and used in subsequent analysis will be referred to as the *optimal objective strategy*. This name is **not** intended to imply that this is the best possible objective strategy. Rather, it is intended to indicate that the goal of this strategy is to find the best possible solution using the least amount of computing resources. This definition is based on the definition of an interpretation problem given in Section 3.

As a basis for defining the optimal objective strategy, let an Optimal Interpretation be defined as follows:

Definition 6.4.1 *Optimal Interpretation,* O *– Given a problem instance,* x*, from an IDP domain definition,* I*, that defines a set,* C*, of connected search spaces corresponding to correct interpretations of* x*, the optimal interpretation,* O*, is such that* $\forall c \in C, O \in C, cost(O) \leq cost(c)$*, where* $cost(c)$ *is the cost of applying the set of operators used to generate* c.

Intuitively, an interpretation can be thought of as a set of operator applications that connects the search space and determines the highest rated explanation for the observed phenomena.

Each of the different elements c of the set C from definition 6.4.1 represents a different set of operators or a different sequence of operator applications. The optimal interpretation is then the set of operator applications that connects the search space with minimum cost, and that always returns a correct answer.

In some situations, it will be necessary to analyze problem solving strategies that are not guaranteed to return the correct answer. To characterize these design parameters of problem solving systems, the notation for *allowed* or *expected error* is defined as, ϵ.

Definition 6.4.2 *Allowed Error, ϵ – For a given IDP, P, the probability that any specific problem solving instance does not return the correct answer.*

Thus, for an IDP with $\epsilon = 0.05$, the problem solver will return the correct answer with probability = 0.95. The other 5% of the time, the problem solver will return an answer that is not the "best" in terms of highest utility (or credibility). ϵ will be used extensively during analysis in situations where a control architecture eliminates certain search paths knowing that they might lead to the correct solution. This might be done in situations where the problem solver recognizes that the likelihood of the path leading to the correct solution is very low and where the differences between elements of a set of highly-rated interpretations is insignificant. In this situation, the problem solver is choosing to trade a limited number of incorrect solutions in order to reduce the expected cost of problem solving.

The definition of Optimal Interpretation will now be restated to include consideration of allowable error:

Definition 6.4.3 *Optimal Interpretation, O – Given an instance of an IDP, P, that defines a set, I, of connected search spaces corresponding to interpretations of P with probability of correctness $= 1 - \epsilon$, the optimal interpretation, O, is such that $\forall i \in I, O \in I, cost(O) \leq cost(i)$.*

An optimal interpretation is now defined to be the set of operator applications that return the correct answer with probability $= 1 - \epsilon$ and that connect the search space with minimal cost. Note that even in situations where the problem solver is allowed a certain amount of error, it is still required to connect the search space. Therefore, any problem solving actions that lead to a non-zero probability of returning an incorrect final solution must have explicit operations that eliminate from consideration some portion of the search space. Thus, correct solutions are in the areas of the search space that have been pruned. (Such a search space, i.e., a search space where the portion of the search space containing the correct solution has been pruned, can be thought of as *overconstrained*.)

In virtually all practical problem solving systems, ϵ plays an important role. In many real-world domains, the cost of exhaustive problem solving is prohibitively expensive. As a consequence, a very common strategy is to simply eliminate certain portions of the search space from consideration at the risk of eliminating the only search paths to a correct interpretation at the same time. Many of the control architectures that will be analyzed with the IDP/*UPC* formalism include allowable errors greater than 0.

6.4.2 Defining an Optimal Interpretation Objective Strategy

We will now use the definition of an optimal interpretation to define an objective strategy that will be used in subsequent analyses. It is important to note that the objective strategy defined here should not be considered the only possible objective strategy. Many other strategies can be defined and incorporated into the IDP/*UPC* analysis framework.

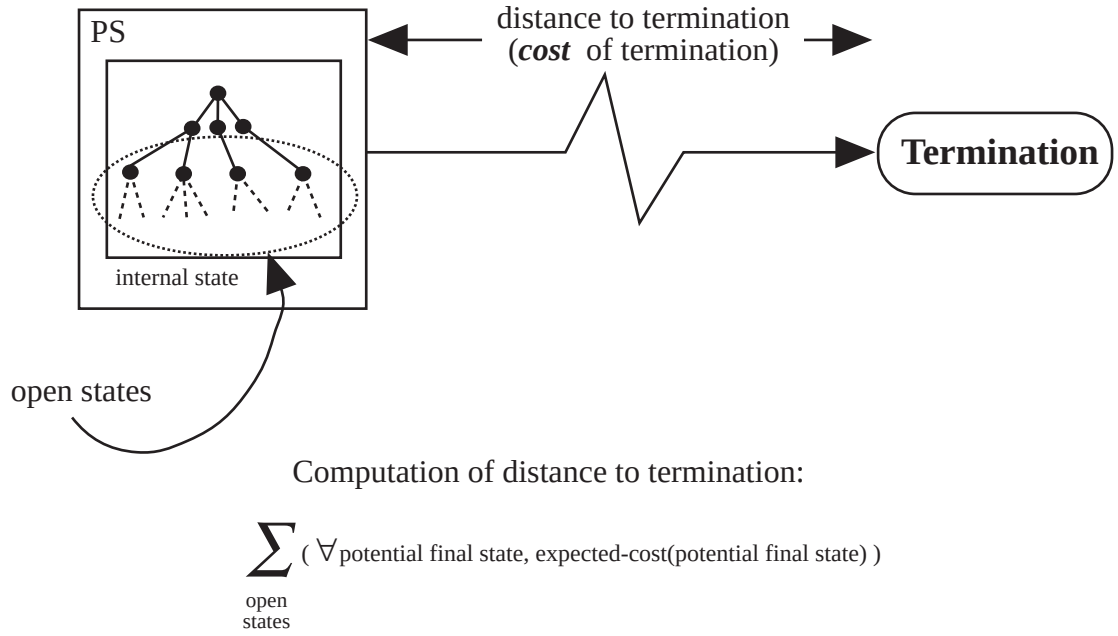


Figure 6.2. Computing the Distance to Termination, C

At any given stage in problem solving, the expected cost to reach termination is equivalent to the expected cost of connecting all open states. This is illustrated in Fig. 6.2 and will be represented as C throughout this thesis. Each step of problem solving reduces C. A given search operator application, op_i, can reduce C in one of two ways. It can succeed in generating new states, which is analogous to traversing search paths in ways that reduce the distance to any final states that can be reached along the paths. Alternatively, the operator can fail, causing all potential paths that required the failed operation to be eliminated from further consideration.

In the first instance, C is reduced to the degree that progress is made in traversing the paths. (The expected cost of path j will be represented as C_j and the degree to which operator op_i reduces C_j will be represented $c_j(op_i)$. The degree to which an operator reduces C will be represented $c(op_i)$.) In the second instance, C is reduced by an amount equal to the expected cost of fully expanding any potential paths that are eliminated.

In convergent search domains, a given operator typically constitutes a segment of multiple paths. In addition, a given operator application might result in the successful extension of some paths, which will be represented as the set E, and the failure of others, which will be represented as the set T. Consequently, the degree to which C is reduced by the application of operator op_i is given by:

Definition 6.4.4 *Amount operator op_i reduces the expected cost to connect all open states, C, is $c(op_i) = \sum_{\forall j \in E} c_j(op_i) + \sum_{\forall k \in T} C_k$, where E is the set of paths extended by the application of op_i and T is the set of paths terminated by the application of op_i.*

Intuitively, each $c_j(op_i)$ represents the degree to which a path to C_j is successfully extended, and each C_k represents the cost reduction associated with the failure of an attempt to extend path k. i.e., in the case of an unsuccessful path extension, the expected cost of problem solving is reduced by the entire distance remaining in path k.

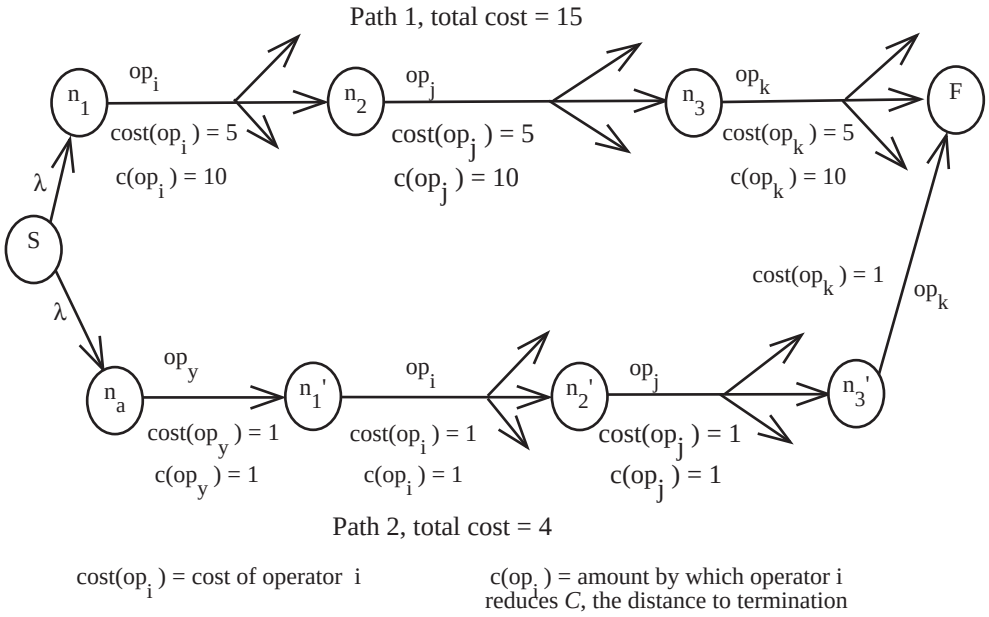


Figure 6.3. Example of the Non-local Effects of an Operator Application

Thus, *by definition*, the optimal objective strategy can be implemented by applying, at each step of problem solving, the operator, op_i, that, for all i, maximizes the average amount of search space connected per unit cost. Written formally, the choice of operator is made in order to maximize:

Equation 6.4.1 $\sum_i c(op_i) / \sum_i cost(op_i)$.

Note that this represents the overall average amount of "search space connected" per unit of problem solving cost. Determining which operator to execute at *each step* of problem solving in order to achieve this maximum may be very difficult, even impossible. This is especially true in situations where the problem solver has only local information about a state. Even so, let us assume, for now, that the optimal objective strategy can be approximated by a control strategy that, at each step of problem solving, chooses the operator that maximizes:

Equation 6.4.2 $c(op_i)/cost(op_i)$.

Equation 6.4.2 is based on a perspective of problem solving that relies solely on *local* information. By local, we mean that this equation only considers the direct effects an operator has on C. It does not take into consideration the effects that an operator may have on other operators that are applied during subsequent processing, e.g., subproblem interactions involving cooperating or competing search paths.

For example, consider the situation shown in Fig. 6.3. In this figure, there are two alternative problem solving paths, one beginning with the state n_1 and the other beginning with the state n_a. In the first, the initial step of problem solving reduces C by 10 units, and it costs 5 units to apply, consequently, $c(op_i)/cost(op_i) = 2$. The other steps along this path also reduce C by 10 units and have cost 5, resulting in similar ratios. (Assume that these costs are based on search paths not shown in the figure.)

In the second path, the initial search step has cost 1, but only reduces C by 1, resulting in a ratio of 1. This step will not be taken until all the operators in the first path have been applied. However, examining the second search path from a more comprehensive perspective results in the observation that the second path subsumes the first and, in fact, is less costly overall.

From a local perspective, it would appear that the first path is a better choice than the second path because, for every operator application in the first path, the ratio from Equation 6.4.1 is greater than the corresponding ratio for the first search step of path 2. However, subsequent steps of the second search path somehow modify the search space so that the space can actually be connected more efficiently than it could in the first path.

This example is somewhat abstract and may not correspond closely to a specific domain, but it serves to illustrate a simple, but very important, principle. This principle is that, from a more comprehensive (or global) perspective, local optima will exist and maximizing Equation 6.4.2 locally will not result in globally optimal interpretations. However, the principles embodied in Equation 6.4.1 can be incorporated with a more comprehensive perspective to define globally optimal objective strategies. Chapter 9 defines the concept of *potential* and discusses how it can be used to overcome problems associated with local optima.

The definition of an optimal interpretation objective strategy will now form the basis for analyzing different control architectures. The analysis technique that will be used will involve formulating the abstractions and approximations used in alternative control architectures as IDP structures and then comparing the results of problem solving based on the grammars.

6.4.3 Local Control Issues - A Brief Discussion

The analytical framework that will be presented in the remainder of this thesis is based on computational methods that are derived relative to a specific control strategy. To be precise, the analytical framework that will be described is based on the control strategy described in this section and extensions to it that incorporate the use of *potential*. It is important to note that, although it is necessary to specify a local control strategy in order to derive the analytical framework, the framework is not dependent on any single control strategy.

It is also interesting to note that in certain situations the choice of local control strategy does not matter. The framework will provide accurate analysis for *any* local control strategy. This will be true in situations where a problem solving system does not incorporate any problem solving actions that prune certain paths based on a dynamic perspective of problem solving. Because our definition of problem solving requires that a problem solver connect the entire search space in order to reach termination, problem solving systems that do not include any pruning operators essentially conduct an exhaustive search. In such a case, the choice of local problem solving strategy is irrelevant.

In situations where the problem solver has access to pruning actions that are based on predetermined criteria, the analysis tools can again be derived independent of the local control strategy. This is because the choice of problem solving activity will have no impact on the pruning operators. The order in which actions are executed will not affect the pruning criteria, and all actions susceptible to being pruned will eventually be subjected to the pruning criteria.

However, in situations where pruning criteria are determined dynamically, the order in which operators are applied is very significant. For example, in an extreme case, if all problem solving actions were applied before any pruning criteria were established, there would be no point to the pruning actions. From an intuitive perspective, it is advantageous to establish pruning criteria early in order to maximize the number of actions that are pruned before they are executed.

Interpretation Grammar G'

0. S → A | B	2. B → DEW
1. A → CD	4. E → jk
3. C → fg	6. W → xyz
5. D → hi	8. j → (signal data)
7. f → (signal data)	10. k → (signal data)
9. g → (signal data)	12. x → (signal data)
11. h → (signal data)	14. y → (signal data)
13. i → (signal data)	15. z → (signal data)

Figure 6.4. Search Operators Defined by Interpretation Grammar G'

6.5 Determining *UPC* Vector Values

In this section, examples of *UPC* vector determination will be presented. These examples will be based on the original interpretation grammar, G', reproduced in Fig. 6.4, and the modified interpretation grammar with added rules for noise and missing reproduced in Fig. 6.7. In this representation, the SNTs of the grammar are A, B, M, N, and O. Note that in this grammar there are no pruning operators, so the computations presented here ignore the computation of the pruning factor, R_{prune}.

6.5.1 *UPC* Vector Values in a Simple Grammar

Figure 6.5 shows the *UPC* values for two low-level states, h and f, from the interpretation grammar shown in Fig. 12.1. The grammar is reproduced in Fig. 6.4 for convenience. In the figure, the subscripts indicate which $SNTs$ the *UPC* values are associated with. For computing the *UPC* values for h, we will assume that h is the only state created so far. We make a similar assumption when computing *UPC* values for f. There is only a single operator available to extend each state and, as a result, there are only single U, P and C vectors for each state, as shown in Fig. 6.5. For state h, the available operator is op_5, which is represented as "D → hi" in Fig. 12.1. For state f, the available operator is op_3, which is represented as "C → fg."

To simplify the computations used in the next two sections, we will limit the discussion of calculating expected utilities. Thus, for all examples in the next two sections, *UPC* values will always reflect an expected utility of 1.

Intuitively, the assumptions used in the next two sections can be thought of as follows. The low-level state h can be used to derive two interpretations, an A or a B. In this example, the utility of either interpretation will be represented as "1." This is manifested in $u_{h,5}$ as two entries, both equal to 1.

By definition, the entries in $p_{h,5}$ correspond to the conditional probabilities of generating paths from state h to each of the final states represented in $u_{h,5}$. Given no prior information about the distribution of domain events corresponding to interpretations of A and B, it will be assumed that the distribution is split evenly between them. Consequently, $P(A) = P(B) = 0.5$. Now the entries in the vector $p_{h,5}$ can be determined.

$u_{h,5} = (1_A, 1_B)$	$p_{h,5} = (0.5_A, 0.5_B)$	$c_{h,5} = ((6,3)_A, (10,3)_B)$
$u_{f,3} = (1_A)$	$p_{f,3} = (1_A)$	$c_{f,3} = ((6,0)_A)$

Figure 6.5. *UPC* Vectors for States from Search Space Defined by G'

$p_{h,5}(1)$ is the probability that a path can be constructed from h, beginning with op_5, to final state A. From Definition 6.3.8,

$$p_{h,5}(1) = P(A \mid h) = \frac{P(A \cap h)}{P(h)}. \tag{6.1}$$

In G', $P(A \cap h) = P(A \rightarrow h)$, since A is not ambiguous with any other interpretations. So, from Definitions 6.3.15, 6.3.13, and 6.3.11,

$$P(A \cap h) = P(A \rightarrow h) = P(A) * P(h \in \{RHS(A)\}^*) = 0.5 * 1 = 0.5. \tag{6.2}$$

Note that in this case, h is on the RHS of D with

$$P(h \in \{RHS(D)\}) = 1, \tag{6.3}$$

and D is in the RHS of A with

$$P(D \in \{RHS(A)\}) = 1. \tag{6.4}$$

Thus,

$$P(h \in \{RHS(A)\}^*) = 1 * 1 = 1. \tag{6.5}$$

From Definition 6.3.12,

$$P(h) = P(h \in \{RHS(A)\}^*) + P(h \in \{RHS(B)\}^*) = 0.5 * 1 + 0.5 * 1 = 1. \tag{6.6}$$

h is included in an RHS of A or B (recursively) with probability 1, and $P(A) = P(B) = 0.5$.

Thus,

$$P(A \mid h) = \frac{0.5}{1} = 0.5. \tag{6.7}$$

This is shown in Fig. 6.5 as $p_{h,5}(1)$.

The computation of $p_{h,5}(2)$, the probability that a path can be constructed from h, beginning with op_5, to final state B, is similar and yields the same result.

Given the utility and probability vectors shown in Fig. 6.5, it is unclear whether the partial interpretation h is part of an A or a B. To differentiate which of the two events occurred, the problem solver must continue to interpret the data by extending partial interpretation h. If the correct interpretation generated from h is an A, then the data corresponding to B's component set (component sets are defined in Chapter 4.4.1) will not be generated unless the data are also in A's component set. Conversely, if the correct interpretation is B, then partial interpretations corresponding to A's component set will not be formed unless they are also in B's component set.

To compute the expected cost vectors, we will let the cost of each production rule be 1. The generation of an A requires that 7 production rule operators be executed, each at a cost of 1. However, the existence of an h implies that the cost of generating an h does not have to be incurred again. Consequently, the expected cost to generate an A, when A is the correct interpretation, is

$$c_{h,5}(1,1) = 7 - 1 = 6. \tag{6.8}$$

Similarly, the expected cost to generate a B, when B is the correct interpretation, is

$$c_{h,5}(2,1) = 11 - 1 = 10. \tag{6.9}$$

In situations where the correct interpretation is a B, the cost of attempting to generate a path to A, starting with op_5, is, by Definition 6.3.19,

$$c_{h,5}(j,2) = \sum_{\forall R} \frac{P(R \rightarrow h)}{(P(h) - P(A \cap h))} * C_{\neg A,R} \tag{6.10}$$

where A is not ambiguous with R.

Therefore,

$$c_{h,5}(1,2) = \frac{P(B \rightarrow h)}{P(h) - P(A \cap h)} * C_{\neg A,B} = \tag{6.11}$$

$$\frac{0.5}{0.5} * (3) = 3. \tag{6.12}$$

$C_{\neg A,B}(h)$, the *expected cost of a failed attempt to generate a path from h to A when B is the correct interpretation,* is the cost of generating a D, since $P(D \mid h) = 1$, plus the cost of attempting to generate an f or g. When B is the correct interpretation, attempting to construct a path to A will fail after a cost of 3 is incurred. This cost will be associated with generating a D (cost of 2) with the application of op_5, which will be successful, and the cost of trying to generate a C, which will be unsuccessful. The attempt to generate a C will fail when the problem solver attempts to generate an f or a g. For now, we will assume that the cost of such a failure is 1. After failing to generate an f or g, we will also assume that the problem solver suspends its attempt to generate a C, and, as a result, its attempt to generate an interpretation of A.

Thus, the expected cost of attempting to generate a path to final state A, given an h, is, from Definition 6.3.16,

$$p_{h,5}(1) * 6 + (1 - p_{h,5}(1)) * 3 = 4.5. \tag{6.13}$$

The computation of $c_{h,5}(2,2)$ is similar and the expected cost of attempting to generate a path to final state B, given an h, is

$$p_{h,5}(2) * 10 + (1 - p_{h,5}(2)) * 3 = 6.5. \tag{6.14}$$

In contrast to h, the low-level state f can only be used to generate an A. Consequently, f's utility vector has only a single entry corresponding to an A. Since there is no other possible interpretation of an f, the probability of reaching the final state associated with an interpretation of A is 1.0. This value can also be computed from Definitions 6.3.8 through 6.3.12. Finally,

1. A → CD	7. f → (signal data)	16. M → Y
2. B → DEW	8. j → (signal data)	17.0 Y → qr
3.0 C → fg	9. g → (signal data)	17.1 Y → qhri
3.1. C → fgq	10. k → (signal data)	18. N → Z
4. E → jk	11. h → (signal data)	19. Z → xy
5.0 D → hi	12. x → (signal data)	20. O → X
5.1. D → rhi	13. i → (signal data)	21.1. X → fgh
6.0 W → xyz	14. y → (signal data)	21.2. X → fg
6.1. W → xy	15. z → (signal data)	

Interpretation Grammar G' with noise and missing data rules

Figure 6.6. G' with Added Noise and Missing Data Rules

the cost vector has a single entry corresponding to the interpretation of A and this is computed as previously described,

$$cost(A) - cost(f) = 6. \tag{6.15}$$

Since a path always exists from state f to an A, the cost of failing to reach A is 0. Consequently, $c_{f,3}(1,1) = 6$ and $c_{f,3}(1,2) = 0$.

Given an h and an f with these *UPC* values, it should be noticed that f is not included in any ambiguous interpretations. As can be seen from f's *UPC* values, this simplifies problem solving greatly. In this domain, given an f, a problem solver can postulate an interpretation of A without conducting any additional problem solving. However, given an h, the problem solver must still differentiate the possible interpretations A and B.

6.5.2 *UPC* Vector Values with Noise and Missing Data

Now consider the *UPC* values for h and f given the domain theory represented by the grammar shown in Fig. 6.6. (This grammar is shown graphically in Fig. 6.7.) This grammar is identical to G' but rules have been added corresponding to noise and missing data. As a result of these rules, the *UPC* values for h and f are as shown in Fig. 6.8. As with the previous example, these values were computed based on the assumption that they were the only states created so far.

Again, final states will be assigned utility 1. op_5 can lead to two final states, A and B. This is indicated in $u_{h,5}(1)$ and $u_{h,5}(2)$, respectively. Likewise, op_{17} can lead to an M and op_{21} can lead to an O. This is represented in $u_{h,17}(1)$ and $u_{h,21}(1)$.

To generate the conditional probability vectors, it was again assumed that the domain events that correspond to the interpretations A, B, M, N, and O are evenly distributed, i.e., $P(A) = P(B) = P(M) = P(N) = P(O) = 0.2$. Furthermore, for this example, the ψ distribution for the possible RHSs of the nonterminals Y, X, C, D, and W will be 0.5. For example, $\psi(\text{Y}\rightarrow\text{qr}) = 0.5$ and $\psi(\text{Y}\rightarrow\text{qhri}) = 0.5$.

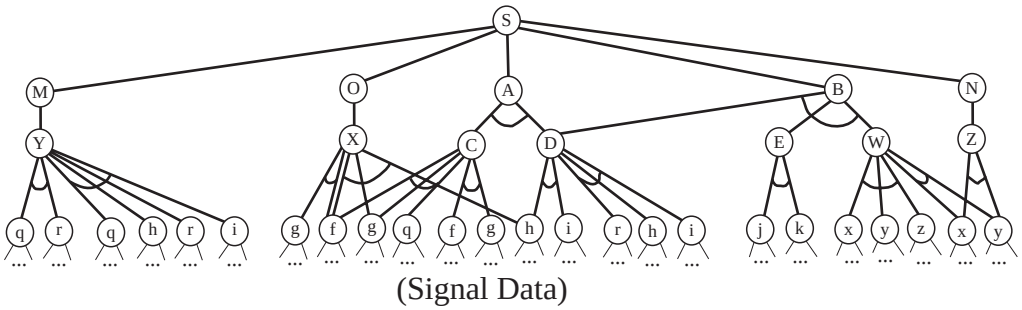


Figure 6.7. Graphical Representation of G'

$u_{h,5} = (1_A, 1_B)$	$p_{h,5} = (0.33_A, 0.33_B)$	$c_{h,5} = ((7,3)_A, (10,3)_B)$
$u_{h,17} = (1_M)$	$p_{h,17} = (0.25_M)$	$c_{h,17} = ((5,1)_M)$
$u_{h,21} = (1_O)$	$p_{h,21} = (0.5_O)$	$c_{h,21} = ((4,1)_O)$
$u_{f,3} = (1_A)$	$p_{f,3} = (0.5_A)$	$c_{f,3} = ((7,3)_A)$
$u_{f,21} = (1_O)$	$p_{f,3} = (1.0_O)$	$c_{f,3} = ((3.75,0)_O)$

Figure 6.8. *UPC* Vectors for Two States from Interpretation Grammar G'

Given this information and from the previous definitions, the probability that a path from h beginning with op_5 can reach final state A is

$$p_{h,5}(1) = \frac{P(A \cap h)}{P(h)}. \tag{6.16}$$

In this grammar, $P(A \cap h) = P(A \rightarrow h)$, since A is not ambiguous with any other interpretations.

$$\begin{aligned} P(A \rightarrow h) &= P(A) * P(h \in \{RHS(A)\}^*) = 0.2 * 1 = 0.2 && (6.17)\\ P(h) &= P(h \in \{RHS(A)\}^*) + P(h \in \{RHS(B)\}^*) + P(h \in \{RHS(M)\}^*) + \\ &\quad P(h \in \{RHS(O)\}^*) && (6.18)\\ &= (0.2 * 1) + (0.2 * 1) + (0.2 * 0.5) + (0.2 * 0.5) = 0.6 && (6.19) \end{aligned}$$

Thus,

$$p_{h,5}(1) = \frac{0.2}{0.6} = 0.33. \tag{6.20}$$

The computation of $p_{h,5}(2)$, the probability that final state B can be reached, is similar and also yields 0.33.

The probability that a path from h beginning with op_{17} can reach final state M is

$$p_{h,17}(1) = \frac{P(M \cap h)}{P(h)}. \tag{6.21}$$

From Definition 6.3.13,

$$P(M \cap h) = P(M \rightarrow h) + P(A \rightarrow \{qrh\}) = \tag{6.22}$$

$$P(M \rightarrow h) + P(A) * P(q \in \{RHS(A)\}^*) * P(r \in \{RHS(A)\}^*) * P(h \in \{RHS(A)\}^*), \tag{6.23}$$

since M is ambiguous with A when the domain event A results in the generation of a q and an r in addition to an h.

$$P(M \rightarrow h) = P(M) * P(h \in \{RHS(M)\}^*) = 0.2 * 0.5 = 0.1 \tag{6.24}$$

and

$$P(A \rightarrow \{qrh\}) = \tag{6.25}$$

$$P(A) * P(r \in \{RHS(A)\}^*) * P(q \in \{RHS(A)\}^*) * P(h \in \{RHS(A)\}^*) = \tag{6.26}$$

$$0.2 * 0.5 * 0.5 * 1.0 = 0.05. \tag{6.27}$$

Thus, using $P(h)$ calculated above,

$$p_{h,17}(1) = \frac{0.1 + 0.05}{0.6} = 0.25. \tag{6.28}$$

The computation of $p_{h,21}(1)$, which is the probability that a path from h beginning with op_{21} can reach final state O, must take into account the fact that the interpretation of an O is always ambiguous with the interpretation of an A, i.e., A $\Rightarrow$ O. (This is true because f and g are always components of $\{RHS(A)\}^*$ and f and g are the terminal symbols that lead to the interpretation of an O.) Thus,

$$P(O \mid h) = \frac{P(O \cap h)}{P(h)} = \frac{P(O \rightarrow h) + P(A \rightarrow h)}{P(h)}, \tag{6.29}$$

$$P(O \rightarrow h) + P(A \rightarrow h) = \tag{6.30}$$

$$P(O) * P(h \in \{RHS(O)\}^*) + P(A) * P(h \in \{RHS(A)\}^*) = \tag{6.31}$$

$$0.2 * 0.5 + 0.2 * 1 = 0.3 \tag{6.32}$$

Thus,

$$p_{h,21}(1) = \frac{0.3}{0.6} = 0.5. \tag{6.33}$$

The expected costs shown for h take into account the distribution function, ψ. When h is included in the derivation of an A or a B, the expected cost varies depending on the distribution of the RHSs for the partial interpretations C, D, and W. The RHSs of D are "hi" and "rhi," the RHSs of C are "fgq" and "fg," and the RHSs of W are "xyz" and "xy." For this example, let

$$\psi(D) = \begin{cases} 0.5 & \text{for D} \rightarrow \text{hi} \\ 0.5 & \text{for D} \rightarrow \text{rhi} \end{cases} \tag{6.34}$$

$$\psi(C) = \begin{cases} 0.5 & \text{for C} \rightarrow \text{fg} \\ 0.5 & \text{for C} \rightarrow \text{fgq} \end{cases} \tag{6.35}$$

$$\psi(W) = \begin{cases} 0.5 & \text{for W} \rightarrow \text{xyz} \\ 0.5 & \text{for W} \rightarrow \text{xy} \end{cases} \tag{6.36}$$

The interpretation trees for A and B that are derived from these rules are shown in Fig. 6.9. There are four distinct interpretation trees for both A and B that include h in their component

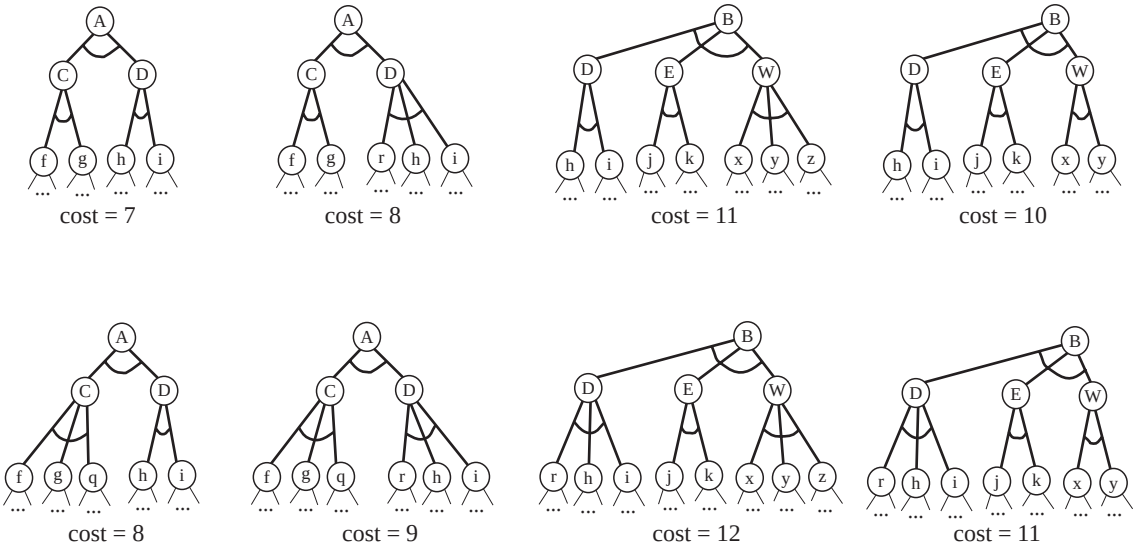


Figure 6.9. Interpretation Trees for Domain Events A and B

sets. In general, such derivations could be semantically different and, as a result, could constitute different final states. In this example, all interpretations of A or B are considered to be represented by the same final state. Thus, there is a single entry in h's *UPC* vectors for a final state corresponding to an A and, likewise, one for a B.

Given the distributions described above, the conditional probability of each of these interpretation trees being correct given an A or a B, respectively, is 0.25 (i.e., $0.5 * 0.5 = 0.25$), and the expected derivation cost for an A from state h is

$$c_{h,5}(1,1) = (0.25 * 7) + (0.25 * 8) + (0.25 * 8) + (0.25 * 9) - 1 = 7. \tag{6.37}$$

Similarly, the expected derivation cost for a B from state h is

$$c_{h,5}(1,1) = (0.25 * 11) + (0.25 * 10) + (0.25 * 12) + (0.25 * 11) - 1 = 10. \tag{6.38}$$

The expected costs when these final states cannot be reached are specified by Definition 6.3.19

$$c_{h,5}(1,2) = \sum_{\forall \boldsymbol{R}} \frac{P(R \rightarrow h)}{(P(h) - P(A \cap h))} * C_{\neg \boldsymbol{A},\boldsymbol{R}} \tag{6.39}$$

where A is not ambiguous with R.

In the case where a path to A cannot be generated and B is correct, op_5 will successfully generate a D, but will fail to generate an f or g. The cost of generating the D is 2, with probability 0.5, or 3, also with probability 0.5. This is specified by

$$\psi(D) = \begin{cases} 0.5 & \text{for D} \rightarrow \text{hi} \\ 0.5 & \text{for D} \rightarrow \text{rhi} \end{cases} \tag{6.40}$$

In the case where the RHS is "hi," the cost of generating a D will be 2; the cost of generating an i plus the cost of generating a D, or $1 + 1 = 2$. In the case where the RHS is "rhi," the cost of generating a D will be 3; the cost of generating an r, plus the cost of generating an i, plus the cost of generating a D, or $1 + 1 + 1 = 3$. The cost of failing to generate g or f is 1. Thus, the cost of failing to generate a path to A given B is

$$0.5 * 2 + 0.5 * 3 + 1 = 3.5. \tag{6.41}$$

In the case where M is true, op_5 will again succeed, as before, and the path will fail when an attempt is made to generate an f or g. However, when M is correct the cost of generating a D is 3. This is due to the fact that h only appears on the RHS of M with an r, a q, and an i. Therefore, the cost of failing to generate a path to A when M is the correct interpretation is

$$3 + 1 = 4.0. \tag{6.42}$$

In the case where O is is true, op_5 will fail, at a cost of 1, and the cost of failing to generate a path to A when O is the correct interpretation is 1.

Therefore, from Definition 6.3.19,

$$\begin{aligned} c_{h,5}(1,2) &= \frac{P(B \rightarrow h)}{(P(h) - P(A \cap h))} * (3.5) + \frac{P(M \rightarrow h)}{(P(h) - P(A \cap h))} * (4) \\ &\quad + \frac{P(O \rightarrow h)}{(P(h) - P(A \cap h))} * (1.0) = \end{aligned} \tag{6.43}$$

$$\frac{0.2}{(0.6 - 0.2)} * (3.5) + \frac{0.1}{(0.6 - 0.2)} * (4) + \frac{0.1}{(0.6 - 0.2)} * (1.0) = \tag{6.44}$$

$$(0.5 * 3.5) + (0.25 * 4) + (0.25 * 1.0) = 3. \tag{6.45}$$

The expected cost of attempting to generate a path to final state A, given an h, is, from Definition 6.3.16,

$$p_{h,5}(1) * 7 + (1 - p_{h,5}(1)) * 3 = 0.33 * 7 + 0.67 * 3 = 4.33. \tag{6.46}$$

Computing the expected cost of attempting to generate a path to final state B, given an h, requires computing $c_{h,5}(2,2)$. The expected costs that will be incurred for failed paths are the same as for A, so

$$c_{h,5}(2,2) = \frac{P(A \rightarrow h)}{(P(h) - P(B \cap h))} * (3.5) + \frac{P(M \rightarrow h)}{(P(h) - P(B \cap h))} * (4) +$$

$$\frac{P(O \rightarrow h)}{(P(h) - P(B \cap h))} * (1.0) = \tag{6.47}$$

$$\frac{0.2}{(0.6 - 0.2)} * (3.5) + \frac{0.1}{(0.6 - 0.2)} * (4) + \frac{0.1}{(0.6 - 0.2)} * (1.0) = \tag{6.48}$$

$$(0.5 * 3.5) + (0.25 * 4) + (0.25 * 1.0) = 3. \tag{6.49}$$

Therefore, the expected cost of attempting to generate a path to final state B, given an h, is,

$$p_{h,5}(2) * 10 + (1 - p_{h,5}(2)) * 3 = 0.33 * 10 + 0.67 * 3 = 5.33. \tag{6.50}$$

Computing the expected cost of attempting to generate a path from h to M, beginning with op_{17};

$$c_{h,17}(1,1) = 5.0 \tag{6.51}$$

There is only one derivation for M, given an h, and it has cost 5.

To compute $c_{h,17}(1,2)$, requires $C_{\neg M,B}$, $C_{\neg M,A}$, and $C_{\neg M,O}$. The value of each of these is 1. The cost of failing to generate an M is 1 – the problem solver tries to generate a Y and fails. In addition, the cost of failing to construct a path to M must take into consideration the fact that M is ambiguous with A when A leads to the generation of an r and a q. Therefore,

$$\begin{aligned} c_{h,17}(1,2) \quad = \quad & \frac{P(A \rightarrow h) - P(A \rightarrow \{qrh\})}{(P(h) - P(M \cap h))} * (1) + \\ & \frac{P(B \rightarrow h)}{(P(h) - P(M \cap h))} * (1) + \frac{P(O \rightarrow h)}{(P(h) - P(M \cap h))} * (1) = \end{aligned} \tag{6.52}$$

$$\frac{0.2 - 0.05}{0.6 - 0.15} + \frac{0.2}{0.6 - 0.15} + \frac{0.1}{0.6 - 0.15} = 1. \tag{6.53}$$

Therefore, the expected cost of attempting to generate a path to final state M, given an h, is,

$$p_{h,17}(1) * 5 + (1 - p_{h,17}(1)) * 1 = 0.25 * 5 + 0.75 * 1 = 2. \tag{6.54}$$

The expected cost of attempting to generate a path from h to O, beginning with op_{21} is computed from;

$$c_{h,21}(1,1) = 4.0 \tag{6.55}$$

There is only one derivation for O, given an h, and it has cost 4.

The cost of failing to construct a path to O given an h must take into consideration the fact that O is ambiguous with A. Furthermore, the cost of failing to generate an O given B or M is 1. (The problem solver fails to generate an X with cost 1.) Therefore,

$$c_{h,21}(1,2) = \frac{P(B \rightarrow h)}{(P(h) - P(O \cap h))} * (1) + \frac{P(M \rightarrow h)}{(P(h) - P(O \cap h))} * (1) = \quad (6.56)$$

$$\frac{0.2}{0.6 - 0.3} + \frac{0.1}{0.6 - 0.3} = 1. \quad (6.57)$$

Therefore, the expected cost of attempting to generate a path to final state O, given an h, is,

$$p_{h,21}(1) * 4 + (1 - p_{h,21}(1)) * 1 = 0.5 * 4 + 0.5 * 1 = 2.5. \quad (6.58)$$

The *UPC* values for state f computed in a similar way, taking into account that f is ambiguous with A, and are shown in Fig. 6.8. Based on these values, the expected cost of attempting to generate a path to final state A, given an f, is

$$p_{f,3}(1) * 7 + (1 - p_{f,3}(1)) * 3 = 0.5 * 7 + 0.5 * 3 = 5. \quad (6.59)$$

The expected cost of attempting to generate a path to final state O, given an f, is

$$p_{f,21}(1) * 3.75 + (1 - p_{f,21}(1)) * 0 = 1.0 * 3.75 + 0 = 3.75. \quad (6.60)$$

6.6 Discussion

The examples in the preceding two sections demonstrate how *UPC* values are calculated for states in an interpretation problem that does not include pruning operators. In addition, they show how costs increase as a result of problem structures associated with noise and missing data. (The effects from masking and distortion are identical.)

For example, compare the costs of *connecting* state h in the first case with similar costs in the second case. As defined in Chapter 3, an interpretation problem solver terminates only after all the search states have been connected. This means that all potential paths have been either explored or pruned. In the first example, h is connected only after the potential paths to A and B have been explored, and the cost of connecting h is 11. This is computed by summing the expected costs of each of the potential paths from h to final states. Specifically, two final states can be reached from h, A and B. The expected cost of the path to final state A is 4.5 (from equation 6.5.1) and the expected cost of the path to final state B is 6.5 (from equation 6.5.1).

In the second case, after the introduction of noise and missing data, the cost of connecting h is 14.15 (from equations 6.5.2 and 6.5.2). Similarly, the cost of connecting state f increases from 6 to 8.75.

In this simple example, the effects of noise and missing data were relatively modest. In fact, they were intentionally kept modest to improve readability. In real-world domains, the effects of uncertainty on the cost of connecting states are much more significant. In some domains, the increase in costs can be polynomial or exponential. An example of such a domain will be given in a subsequent section.

These two examples demonstrate a very important observation that will be addressed in the remainder of this thesis. This observation is related to the *way* in which problem solving costs increase. Cost increases can be divided into three categories,

False Positives – One of the causes for increased problem is the result of ambiguity. Specifically, in the second example, summarized in Fig. 6.8, the sum of the probabilities of reaching one of four final states, A, B, M, or O, from h is greater than one! This means that for some inputs, more than one final state can be reached. By definition, in interpretation problems, all final states that can be generated *must* be generated and their utilities compared in order to determine which is the correct interpretation. Therefore, when there are multiple competing final interpretations, the work associated with generating the incorrect, or "false," interpretations is wasted.

False Negatives – Another cause for increased problem solving costs is the result of semi-ambiguity, i.e., partial paths that require a non-zero amount of work to eliminate. We refer to these phenomena as "false negatives" because they are incorrect search paths that do not lead to complete interpretations of the data. In the examples presented here, the effects of these false negatives can be seen in the cost vectors in Fig. 6.8. In the vectors shown, the costs of attempting to generate a path to a particular final state when the path cannot be generated have increased, especially for state f, over the costs shown in Fig. 6.5.

Redundancy – One of the more significant causes of increased problem solving costs is that associated with ambiguity that results in multiple search paths leading to the same final state. Though not demonstrated explicitly here, these phenomena are similar to the False Positive category, but they are also applicable to correct solutions. In a redundant domain, there may be multiple search paths to a correct interpretation. The effect on the expected cost of problem solving is similar to the effects of False Positives.

6.7 Quantitative Effects of Structural Interaction

The preceding sections formalize the computation of *UPC* values, based on the component, credibility, and cost structures of an IDP$_i$ grammar. In general, there will be a set of values associated with each of the basic IDP structures, a set of probability values associated with the component structure, a set of credibility (or utility) values associated with the credibility structure, and a set of cost values associated with the cost structure. In certain domains, a given set of *UPC* values may be associated with more than a single IDP structure. This phenomenon is referred to as *structural interaction.* Thus, structural interaction occurs when the *UPC* values for an IDP domain are computed based on the interaction of two or more basic structures. For example, when the probability values are computed based on the interaction of the component and credibility structures.

To understand structural interactions, consider the IDP domain structure shown in Fig. 6.10. This hypergraph is very similar to the one shown in Fig. 3.4, with the notable exception of the 'prime' states. In this example, the prime states correspond to dynamic pruning operators. (Dynamic pruning operators are discussed at great length in Chapters 4 and 11.) A dynamic pruning operator of the form $s' \rightarrow s$ has a corresponding interpretation operator which functions as follows: "given an s, generate an s' if the credibility rating of s is above a certain threshold, t." This interpretation constitutes a structural interaction because the probability of generating an s' is a function not only of the component structure of a domain, but the credibility structure as well. Specifically, the probability of generating an s' given an s is a combination of the conditional probability function, P($s' \mid s$), which is determined from the

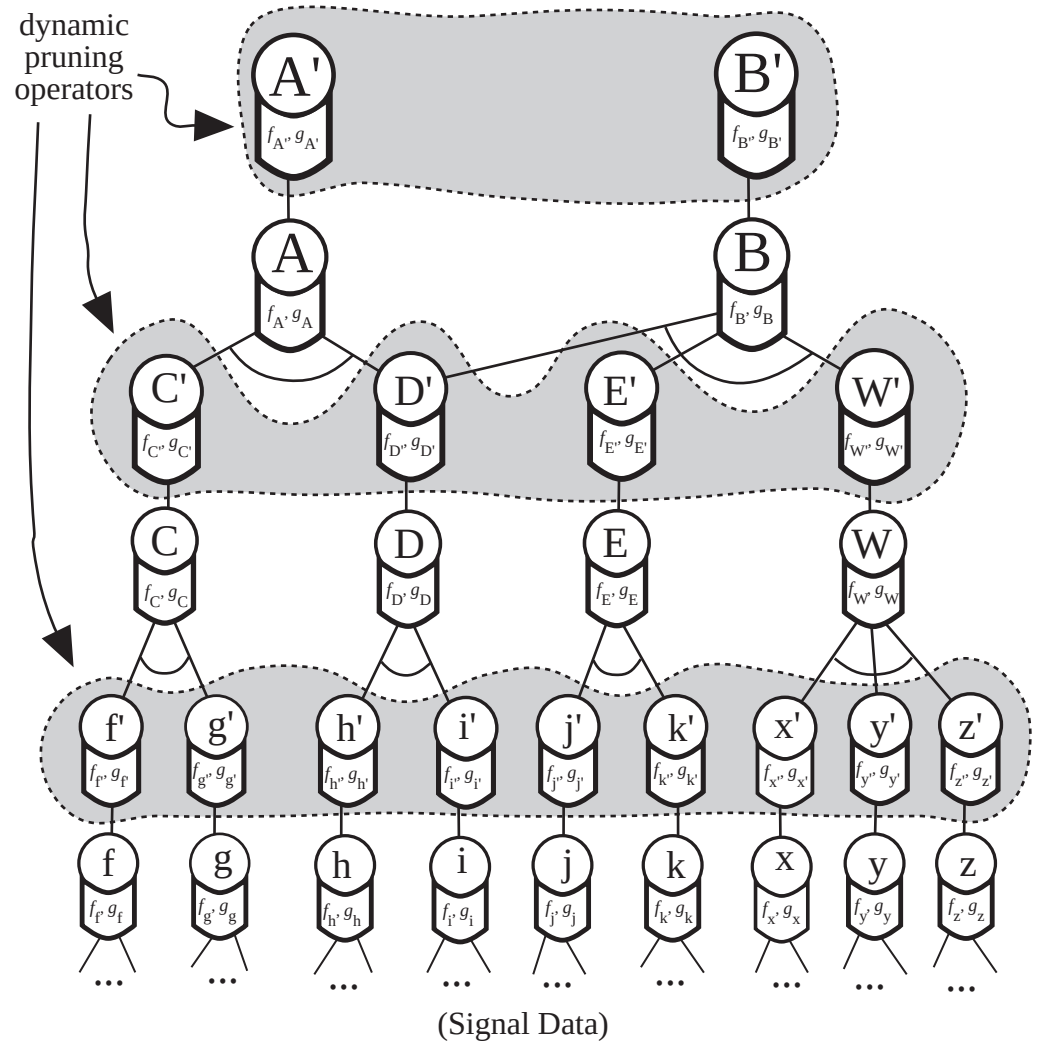


Figure 6.10. Example of the Structural Interaction

component structure, and the distribution of the credibility functions of the grammar. Thus, the component and credibility structures interact.

For example, consider a domain where the pruning threshold is 0 (i.e., no pruning takes place), and the IDP grammar rules are those shown in Fig. 6.10. In this domain, the probability of generating an x' given an x is 1. This is determined from the conditional probability P(x'| x) which is computed from the component structure of the grammar. Now consider the same situation with a pruning threshold of t. In this new situation, the probability of generating an x' given an x is 1 (the conditional probability of generating an x' given an x) multiplied by the probability that the credibility of x is greater than, or equal to, t, or 1 - P(credibility(x) < t).

It is important to note that in domains with structural interactions, the computation of *UPC* values must be done dynamically. This is because the *UPC* values for a state are a function of the characteristics of a state, such as credibility, that are unknown until the state is actually created. Therefore, the *UPC* values cannot be computed a priori. However, it should be noted that certain computations can be done a priori, as will be discussed in Chapter 5.

These computations can be stored and retrieved at run-time to reduce the cost of computing *UPC* values dynamically.

6.8 Chapter Summary

The *UPC* formalism is defined in this chapter. It provides a representation of a search space that can explain and predict the behavior of a search control mechanism. In the *UPC* representation, the traditional concept of a search space state is extended to include vectors indicating a state's location in a search space relative to final states in terms of the cost and probability of reaching the final state and the final state's expected utility. The *UPC* formalism can be thought of as computational structure based on statistical characteristics of IDP models that can be used to simulate an optimal problem solving strategy based on IDP statistics and a specification of problem solving operators. Using the *UPC* representation, we can construct problem solving systems capable of achieving the levels of performance predicted by quantitative analysis of IDP domain specifications and the *optimal interpretation control strategy*. The optimal interpretation control strategy is defined in Chapter 6.4. This is a necessary component of an analytical framework because it provides a basis for experimental control, comparison and evaluation. For example, in the experiments in Chapter 7, the base-line used in the experimental comparisons is the performance of a problem solver that uses the optimal interpretation control strategy to evaluation operators. As a consequence, domains with different structures can be compared using identical evaluation function based control architectures or these architectures can be varied to compare performance of different problem solvers within a given domain.

CHAPTER 7

EXPERIMENTAL VERIFICATION OF THE BASIC FRAMEWORK

To verify the analysis framework presented in the preceding chapters, an IDP/*UPC* problem solving testbed was constructed. The basic structure of the testbed is shown in Fig. 5.1. A domain problem structure is specified in the form of a phrase-structured grammar with associated distribution, utility (credibility), and cost functions corresponding to each rule of the grammar. In addition, the problem solver's model of the problem domain is also specified as a phrase-structured grammar with associated functions.

The Domain Simulator uses the specification of the problem domain to generate problem instances. The problem solver uses its model of the problem domain's structure to interpret each problem instance. The problem solving actions available are specified as production rules of the grammar. In addition, each production rule of the grammar has a credibility function associated with it that is used to generate ratings of intermediate and final problem solving states.

The objective strategy that constitutes the basic control component of the problem solver is a simple *best-first* algorithm that attempts to generate an optimal interpretation based on equation 6.4.2. In our problem solving system, credibility is calculated dynamically, as defined in Chapter 6.3, and this calculation is used to determine *UPC* values. The conditional probabilities and expected cost components of the *UPC* vectors are computed a priori. The domain characteristics that change from run to run are represented with the feature list convention [Gazdar *et al.*, 1982, Knuth, 1968].

Using this testbed, we have conducted two sets of verification/validation experiments using the grammars shown in Figures 7.1 and 7.2. The first set of experiments were designed to verify the basic probability and cost estimation functions from preceding chapters and used simplifying assumptions about the utility structure of the domain. The second group of experiments focused on the effects of using incorrect models. In these experiments, the problem solver's model of the actual problem domain's structure is distorted in a variety of ways. The intent was to investigate the manner in which a problem solver's performance is affected by a deviation from an ideal domain theory. The two sets of experiments are summarized in the next two sections.

7.1 Experiment Set 1

The results of the first set of verification experiments are shown in Table 7.1. The column labeled "Grammar" indicates the domain and problem solver specification used in the experiment. $E(Cost)$ indicates the expected cost of problem solving in the domain based on an analysis of the grammar. "Avg. Cost" shows the actual average cost of problem solving, for 100 samples (each of 50 problem solving instances), in the domain. The "Sig" column indicates whether any difference in the Expected Cost and the Average Cost is statistically

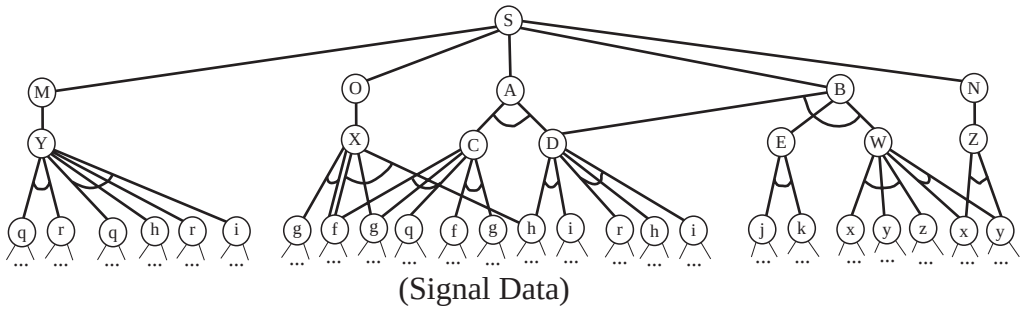


Figure 7.1. Interpretation Grammar G' with Added Noise and Missing Data Rules

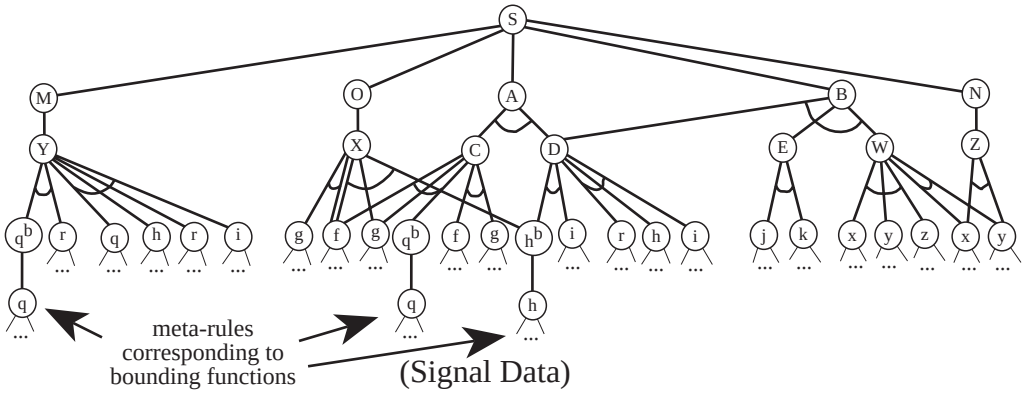


Figure 7.2. Example of Bounding Function Incorporated in a Grammar

significant[1] based on a t test with a 95% confidence level. In each experiment, the hypotheses that are tested are $H_0 : \mu_{cost} = E(C)$ and $H_1 : \mu_{cost} \neq E(C)$ where μ_{cost} is the normalized population mean (each element of the population consisting of 50 problem solving instances), and $E(C)$ is the analytically predicted normalized population mean[2]. For experiments labeled significant, the null hypothesis, H_0, is rejected. The last column shows the number of correct answers that were found.

In these experiments, the grammar employed a credibility structure that simplified the hand computations used to verify the experimental results. In the simplified structure, productions corresponding to noise and missing data in the environment had the same credibility as "correct" interpretations. There was no credibility reduction associated with missing data or noise. As will be seen, this caused the results of credibility based pruning experiments to be somewhat disappointing, since the correlation between low-credibility partial results and correct interpretations was the same for both "correct" partial results and "incorrect" partial

[1]Note that the calculation of significance does not include any consideration of the percentage of correct answers found.

[2]In order to use the t statistic, it is necessary to use variables with normal distributions. The expected cost of a *single* problem solving instance in the grammars we test is not distributed normally. However, by the *Central Limit Theorem*, we can approximate a normal distribution by defining each element of the population to be 50 problem solving instances. Thus, each experiment consisted of 100 samples and each sample included 50 problem solving instances.

results. Experiments in the next section will investigate the effects of pruning in domains where the correlation between low-credibility partial results and "correct" interpretations is lower for "incorrect" partial interpretations. In these domains, a problem solver can benefit significantly by pruning low rated data, i.e., the expected cost of problem solving will decrease significantly but the expected percentage of correct answers does not decrease dramatically.

7.1.1 Experiments 1, 2 and 3

In Experiment 1, every problem solving task, including pruning tasks, was assigned a constant cost of 10. This value was chosen to simplify the verification process. In general, the experimental testbed allows tasks to have arbitrarily complex cost functions. Experiment 2 was similar to Experiment 1 except that bounding functions were added. The bounding functions also had a cost of 10 and eliminated from consideration paths with expected credibilities less than 0.5.

In Experiment 2, the use of a bounding function reduced the cost of problem solving, but it also decreased the number of correct answers found by the system. This is because the bounding functions eliminated certain paths, which reduced the cost, but some correct paths as well as some incorrect paths were included in the set of eliminated paths.

In each of the first three experiments, the parameters of the environment grammar, specifically the distribution functions, were the same as those in the problem solver's grammar.

7.1.2 Experiments 2 and 3

In Experiments 2 and 3, the bounding functions used were simple "threshold cutoffs." If a state had a rating below the threshold, T, the state would not be generated. In these experiments, the credibility structure of the problem solver's grammar was identical to that of the environment grammar used to generate problem instances.

In Experiment 3, the cost of the bounding functions was reduced to 1. This reduced the cost of problem solving more, but it had no effect on the number of correct answers that were found. This is because the same paths were pruned in Experiment 3 as in Experiment 2, the cost reduction was associated with the reduction of the cost of executing a bounding function.

As shown by the percentage of correct answers found in these experiments, the form of bounding function used in these experiments is probably not appropriate. As was discussed above, this is because the bounding functions are pruning both "correct" and "incorrect" interpretations indiscriminately.

7.1.3 Experiments 4 and 5

In Experiments 4 and 5, the problem solver's grammar was unchanged, but the distribution parameters in the environment grammar were altered. In Experiments 1, 2, and 3, the distribution values were always split evenly between all possible alternatives. In Experiments 4 and 5, the distributions were changed to make certain alternatives more likely. In these experiments, for each grammar rule, the more credible right-hand-side (rhs) was assigned a distribution value of 0.9 and the other possible RHSs split the remaining 0.1 evenly.

Table 7.1. Results of Verification Experiments – Set 1

Exp	Generation				Interpretation				Sig	% C
	G	Dist	U	$E(C)$	G	Dist	U	Avg. C		
1	1	even	0.5	201	1	even	0.5	203	N	100
2	2	even	0.5	189	2	even	0.5	187	N	80
3	3	even	0.5	180	3	even	0.5	181	N	80
4	1	skew	0.5	368	1	even	0.5	369	N	100
5	3	skew	0.5	198	3	even	0.5	198	N	96
6	3	even	0.25	157	3	even	0.5	156	N	50
7	3	even	0.75	194	3	even	0.5	193	N	92

Abbreviations

Exp: Experiment
Generation: Description of IDP Domain Grammar used to generate problem instances.
Interpretation: Description of IDP Interpretation Grammar specifying the problem solver.
G: The IDP grammar used;
1: G'
2: G' and bounding functions with cost 10,
3: G' and bounding functions with cost 1,
Dist: Distribution of Domain Events;
even: domain events evenly distributed
skew: distribution skewed to more credible events,
U; expected problem instance credibility;
0.5: problem instances have expected credibility 0.5
0.25: problem instances have expected credibility 0.25
0.75: problem instances have expected credibility 0.75
$E(C)$: Expected Cost of problem solving for given grammar
Avg. C: actual average cost for 100 samples of 50 random problem instances each
Sig: Whether or not the difference between expected cost and
the actual average cost was statistically significant
Y: yes, there is a statistically significant difference
N: no, there is not a statistically significant difference
% C: percentage of correct answers found

7.1.4 Experiments 6 and 7

In Experiments 6 and 7, the distribution functions used by the environment grammar and problem solver's grammar were identical, but the credibility generation functions differed. In the previous experiments, the credibility generation functions were based on a random function that produced an "average normalized credibility" of 0.5. The bounding function used would prune a state "q" or "h" with a rating less than 0.3.

In Experiments 6 and 7, the same bounding functions were used, but the credibility generation functions were changed in the environment grammar. In Experiment 6, the credibility function was altered to generate "average normalized credibilities" of 0.25. In Experiment 7, they generated "average normalized credibilities" of 0.75.

In Experiment 6, this lead to lower average problem solving costs, compared with experiment 3, as many paths were pruned, and fewer correct answers, as many of the pruned paths were actually correct paths. This was consistent with expectations. By lowering the expected credibility of partial results and by retaining the same bounding threshold, many more paths are pruned. Again, since the problem solver cannot differentiate between "correct" and "incorrect" partial interpretations based on their credibilities, the pruned paths included large numbers of both "correct" and "incorrect" paths. This resulted in the shown reduction in correct answers found.

In Experiment 7, the opposite was true. Problem solving costs increased, compared with experiment 3, as fewer paths were pruned, but fewer correct paths were pruned and the percentage of correct answers increased.

7.2 Experiment Set 2

In this set of experiments, the simple credibility function used in the first set of experiments was replaced with more intuitively correct credibility functions. Using these credibility functions, productions from the grammar associated with noise and missing data generate lower credibilities than those generated by "correct" productions. This change had a significant effect on the experimental results, as shown in Table 7.2.

All of these experiments were based on an expected credibility of 0.5 and a bounding function with an a priori threshold of 0.33. Thus, certain partial results with credibilities lower than 0.33 are pruned.

In contrast to the first set of experiments, in this set of experiments, the correlation between a partial interpretation with a low credibility and a correct solution is much lower for "incorrect" partial results than for "correct" partial results. This is reflected in the results in the general improvement in overall problem solving performance, i.e., the expected cost of problem solving decreases and the percentage of correct answers increases.

In Experiment Set 2, Experiments 8, 9, and 10 are the baselines that other experimental results are compared with. Specifically, Experiments 9 - 17 can be compared with Experiment 8 to observe the effects of altering the distribution functions used by the problem solver so that they are different from those used to generate problem instances and of altering credibility functions in a similar fashion. Experiments 9 and 10 show only the effects of altering the distribution functions. Experiments 11 and 12 are restricted to demonstrating the effects of altering the credibility functions used by a problem solver.

To best judge the effect of modifications to the distribution functions or the credibility functions, Experiments 13, 14, and 17 should be similarly compared with the baseline established in

Experiment 9. These experiments all use the same grammar, but different credibility functions. The credibility functions used in Experiment 9 are correct and Experiments 13 and 14 show the effects of using incorrect credibility functions that either overestimate or underestimate the credibility of results generated with noise and missing data rules. In Experiment 17, the credibility functions used are simply "bad." They rate some partial results too high, and some too low. Experiments 15 and 16 should be compared with the baseline established in Experiment 10.

7.2.1 Experiment 8

Experiment 8 serves as the baseline for the second set of experiments. In this experiment, the problem solver's model of the domain structure is exactly the same as that used to generate problem instances. The bounding functions used still prune some correct paths, but not as many as in the first set of experiments.

7.2.2 Experiment 9

In this experiment, the distribution of noise and missing data in the domain was increased, but the model used by the problem solver was not changed. The increase in noise caused a small increase in the cost of problem solving when compared with Experiment 8. Also, more correct paths were mistakenly pruned. This was due to the fact that increasing the distribution of noise and missing data decreased the number of higher rated correct problem instances. Consequently, more correct paths had lower ratings and were mistakenly pruned.

7.2.3 Experiment 10

The distribution of noise and missing data in the domain was decreased, and again the model used by the problem solver was not changed. There was a slight increase in the cost of problem solving related to fewer paths being pruned, but there was also a decrease in cost associated with fewer and less expensive incorrect paths, especially those associated with noise. These two effects canceled each other out and the difference between the cost of problem solving in this experiment and the cost from Experiment 8 was small. There was an increase in the percentage of correct answers found. This resulted from fewer paths being pruned. i.e., given that there was less noise and missing data, fewer "bad paths" were explored. This lead to a reduction in the cost of problem solving and an increase in the number of correct answers found.

7.2.4 Experiment 11

The problem solver's model of the distribution of domain events was the same as that used to generate problem instances, but the problem solver's model of credibility rated partial results generated from missing data lower than it should have. The effects of this were beneficial. In a sense, the problem solver rated "bad" data lower than it should have. The effect was that the problem solver pruned the bad data more often and reduced the overall cost of problem solving compared to Experiment 8. Unfortunately, the problem solver also rated some "good" data lower than it should have, and it pruned this as well, resulting in fewer correct answers found.

Table 7.2. Results of Verification Experiments – Set 2

Exp	Generation				Interpretation				Sig	% C
	G	Dist	U	$E(C)$	G	Dist	U	Avg. C		
8	3	even	0.5	191	3	even	0.5	190	N	96
9	3	skew1	0.5	195	3	even	0.5	195	N	91
10	3	skew2	0.5	189	3	even	0.5	190	N	98
11	3	even	0.5	179	3	even	low	179	N	94
12	3	even	0.5	203	3	even	high	202	N	98
13	3	skew1	0.5	191	3	even	low	191	N	88
14	3	skew1	0.5	209	3	even	high	210	N	98
15	3	skew2	0.5	185	3	even	low	185	N	95
16	3	skew2	0.5	198	3	even	high	199	N	98
17	3	skew1	0.5	203	3	even	bad	202	N	92

Abbreviations

Exp: Experiment

G: The problem solving grammar used;
1: G'
2: G' and bounding functions with cost 10,
3: G' and bounding functions with cost 1,

Dist: Distribution of Domain Events;
even: domain events evenly distributed
skew1: distribution skewed to more noise and missing data
skew2: distribution skewed to less noise and missing data

U; expected problem instance credibility;
0.5: problem instances have expected credibility 0.5
low: problem solver rates "bad data" lower
high: problem solver rates "bad data" higher
bad: problem solver rates some "bad data" higher, and some "good data" lower

$E(C)$: Expected Cost of problem solving for given grammar

Avg. C: actual average cost for 100 samples of 50 random problem instances each

Sig: Whether or not the difference between expected cost and the actual average cost was statistically significant
Y: yes, there is a statistically significant difference
N: no, there is not a statistically significant difference

% Correct: percentage of correct answers found

7.2.5 Experiment 12

Again, the problem solver's model of the distribution of domain events was the same as that used to generate problem instances, but the problem solver's model of credibility rated partial results generated from missing data higher than it should have. This resulted in the problem solver pruning fewer partial results, both "correct" and "incorrect." Consequently, the cost of problem solving is higher than in Experiment 8.

7.2.6 Experiment 13

In this experiment, the problem solver's model of the distribution of domain events and credibility are both incorrect. The domain actually generates more missing data and noise than the problem solver expects, and the problem solver rates intermediate results based on the missing data too low. The cost of problem solving increases slightly because there is more noise to process, but because some intermediate results are rated lower, more pruning occurs. The net effect on cost is insignificant when compared with Experiment 8. However, when compared with Experiment 9, which is a more appropriate baseline, the cost of problem solving here is lower. This is because both 9 and 13 generate more noise and missing data than 8, but in 13, the noise and missing data is pruned more often. The pruning, however, is often of "correct" partial results, and the number of correct answers decreases when compared to both Experiments 8 and 9.

7.2.7 Experiment 14

In this experiment, the domain actually generates more missing data and noise than the problem solver expects, and the problem solver rates intermediate results based on the missing data too high. The result is an increase in cost compared with both Experiments 8 and 9. There is a greater amount of expensive noise to process, and the problem solver prunes fewer intermediate results. However, fewer correct paths are pruned resulting in a higher success rate.

7.2.8 Experiment 15

The domain generates less noise and missing data than expected, and the problem solver rates partial results based on missing data lower than it should. This results in a decrease in the cost of problem solving when compared with either Experiment 8 or the more appropriate baseline, Experiment 10. More correct results are pruned resulting in a lower number of correct answers being found. The difference is greater when compared with 10 than with 8.

7.2.9 Experiment 16

The domain generates less noise and missing data than expected, and the problem solver rates partial results based on missing data higher than it should. This results in a slight decrease in the cost of problem solving resulting from less noise to process, but fewer paths are pruned and the net effect is an increase in the cost of problem solving. The increase is significant when compared with either experiment 8 or 10. However, since fewer paths are pruned, the number of correct answers increases.

7.2.10 Experiment 17

In this last experiment, the domain generates more noise and missing data than expected, and it rates partial results based on noise higher than it should. In addition, the problem solver rates correct partial interpretations lower than it should. As a consequence, the cost of problem solving increases due to the increase in the amount of noise that must be processed and fewer correct answers are found, as more correct paths are pruned.

7.3 Chapter Summary

Of the preceding experiments, 17 is probably the most realistic. The results from experiments 8 through 16 are generally consistent with the results from experiments 1 through 7. However, experiments 12 through 16 are more representative of real world domains where a problem solver's model of a domain's structure is slightly off both in terms of modeling the distribution of noise and missing data and in terms of modeling the distribution of credibility.

These domains are all unrealistic in the sense that *all* rules of a particular type were treated the same. For instance, if one missing data rule was set to generate credibility that was too high, then they all were set to generate credibility that was too high. In real domains, it is probably the case that some rules overestimate the correct credibility ratings and some underestimate. The same can be said for a problem solver's model of distribution of missing data and noise.

Experiment 17 is more realistic in the sense that we expect the inaccuracies in a problem solver's model of a domain to both increase the cost of problem solving and to decrease the number of correct answers produced.

These experiments are somewhat limited by the grammar used. The grammar used in experiments 1 through 17 was chosen to preserve consistency with previous chapters of the thesis. It should be remembered, however, that this is a simple grammar that does not represent the full effects of noise and missing data, not to mention distortion effects.

More generally, these experiments statistically verify that the closed form equations used to determine expected costs analytically are correct. The experiments also show that when the domain model is manipulated in various ways, the IDP/*UPC* analysis results conform to intuitive expectations.

CHAPTER 8

EXTENDING THE *UPC* FORMALISM

This section presents an extension to the *UPC* formalism that will model the abstract and approximate reasoning strategies in interpretation problems. These modeling techniques will be particularly useful for formulating meta-operators used in IDPs (defined in Chapter 4.4) as part of a search problem. This will include meta-operators used implicitly by the control component. The emphasis of this section will be on defining the extensions. Subsequent chapters will discuss the implications these extensions have for problem solving strategies.

It is important to stress that the IDP formalism and the *UPC* formalism are independent methods for formalizing the character of a problem domain. In the IDP/*UPC* framework, the two formalisms are linked because the IDP formalism is used to generate *UPC* values. In general, this does not have to be the case. The IDP formalism can be used to analyze problem domains independent of the use of the *UPC* formalism and vice-versa. For other domains, it may not be as easy to represent base-level and meta-level operators in a unified representation as has been done with interpretation domains in this thesis.

In extended *UPC* models, the problem solver has the option to *project* the base search space to a new, abstract search space where it can efficiently solve a simplified version of the problem. However, it is possible that the solution found in the projected space may not be of an acceptable form, so it is *mapped* back to the base search space. In general, this method will improve the overall efficiency of problem solving if, as a result of mapping the solution (or partial result) from a projected space back to the base space, problem solving in the base space is constrained in some way. For example, a partial result from a projected space can provide a more global perspective that can be used by problem solving activities in the base search space.

The extension to the *UPC* formalism is represented in Fig. 8.1. In the implied paradigm, the problem solver has the option to project the base space and conduct processing in the abstract space, mapping results back to the base search space.

Chapter 8.1 presents background material regarding the development and use of projection spaces. Chapter 8.2 presents a formal definition of projection space extensions to *UPC* models.

8.1 Related Research

The effort to formally incorporate notions of projected or abstracted search spaces into the traditional model is based on *Approximate Processing* concepts described in [Erman *et al.*, 1980, Lesser and Pavlin, 1988, Lesser *et al.*, 1988b, Decker *et al.*, 1990] and on *goal processing* concepts described in [Lesser *et al.*, 1989b]. Approximate processing is based on exploiting the structure of a search space to form abstractions of the space with well understood effects. A problem solver capable of exploiting approximate processing has access to "simplified" operators that it can use to search the abstracted version of a given search space. The results of searching the abstract space can be mapped back to the original space and used to enhance problem solving in that space. Experiments with approximate processing [Lesser *et al.*, 1988b,

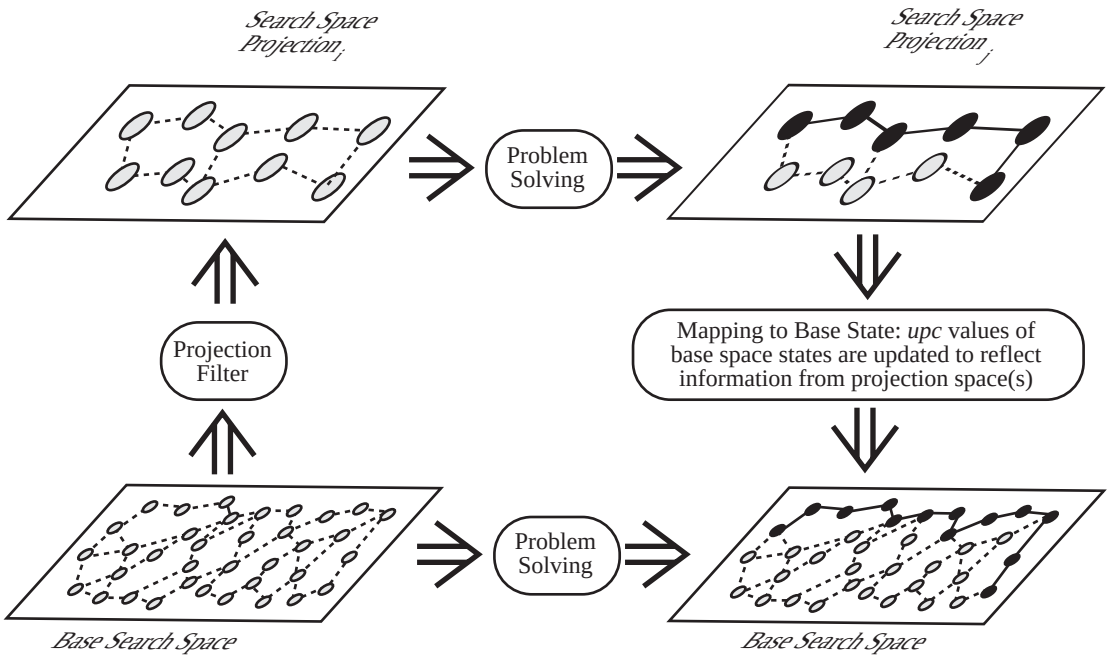


Figure 8.1. The Search Paradigm Implied by the Extended *UPC* Formalism

Decker *et al.*, 1990] showed that significant efficiencies can be gained with careful exploitation of approximate processing mechanisms.

This paradigm can be viewed as a form of hierarchical problem solving, such as that discussed by Newell [Newell *et al.*, 1962], Minsky [Minsky, 1963], and Knoblock [Knoblock, 1991b].

8.2 Formalizing Projection Spaces

The extended *UPC* formalism is intended to explicitly represent characteristics of search spaces that are used implicitly in control architectures. Specifically, the new formalism explicitly represents problem structures defined by subproblem relationships in a way that enables a problem solver to exploit them. (For IDPs, the structures that will be exploited by abstract processing are defined in Chapter 4.4.) The new characterization of a search problem is based on the four-tuple $< \mathcal{S}, \Omega, \omega, \Phi >$, where;

$\mathcal{S}$ = the *start state*; $\mathcal{S}$ is defined by the input data to the problem solver and the initial values of any relevant CVs.

Ω = the *base search space* with associated CVs and operators. Ω corresponds to the traditional notion of a search space. It is defined by CVs that specify the characteristics of individual states (including the *UPC* vectors), operators that map one state to another, and functions of CVs that define final states. In terms of an IDP, Ω is defined by the interpretation grammar.

ω = a set of *projections*, or *abstractions*, of the base search space, each with their associated CVs and operators. Final states are those from the base space that can be reached via mapping

operators. A given search space projection ω_i is defined by two sets of operators, $OP_{(\Omega,\omega_i)}$ and $OP_{(\omega_i,\omega_i)}$. $OP_{(\Omega,\omega_i)}$ is the set of operators that map states from Ω to states in ω_i. $OP_{(\omega_i,\omega_i)}$ is the set of operators that map states in ω_i to other states in ω_i. As with Ω, each state in a projected search space is characterized by a set of CVs including a set of *UPC* vectors.

An appropriate metaphor for a projection of a search space is that it is transformed by "projecting" it through a filter defined by the set of operators $OP_{(\Omega,\omega_i)}$. The result is a blurring of the original space into a simpler, less clearly defined search space. With a well designed filter, states with similar properties will be merged into abstract states and relationships among states will cause sought after states to become more apparent and states with undesirable properties to be eliminated entirely.

Ideally, the results of projection will be a space that is several orders of magnitude less costly to search and that can be mapped back to the original space in a way that reduces problem solving costs. The strategy then is to find a solution (or partial solution) in the projected search space and to somehow use this solution as a guide to problem solving in Ω. If the cost of finding a solution in a projected search space is less than the cost saved, then the strategy is beneficial.

Φ = a set of ***mapping functions*** from projection spaces back to the base search space. The objective of problem solving in an abstract projection space is the generation of constraints that can somehow be used to restrict problem solving in the base search space. Functions in Φ can be thought of as the mechanisms that map constraints from an abstract space, ω_i, back to the base search space, Ω. This can be done by creating new states in Ω, or by modifying existing states. A taxonomy of mapping strategies is defined in Appendix B.

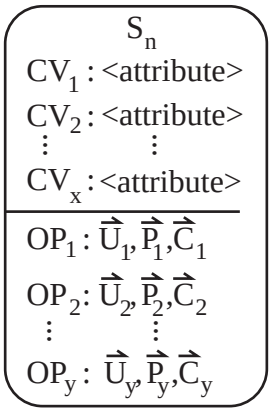
Given these definitions of $< \mathcal{S}, \Omega, \omega, \Phi >$, the *UPC* formalism can now be represented as shown in Fig. 8.2. Figure 8.2.a summarizes the representation of a state. A state in the search space is defined by a set of CVs, where each CV_i is a characteristic variable with a corresponding attribute value. In addition, each state has a set of *UPC* vectors corresponding to each operator available to extend the state. It is important to note that, even though *UPC* vectors only include information corresponding to paths to final states in the base space with non-zero utility, the *UPC* vectors of a state will include information corresponding to *all* paths, even paths that traverse one or more projection spaces, extending from the state. The *UPC* representation will unify the base search space and all relevant projection spaces.

Figure 8.2.b is a representation of how the *state of the problem solver,* S_{PS}, is defined in the *UPC* formalism. In essence, S_{PS} solver is defined by the status of problem solving in the base search space, Ω, and all the projection spaces, ω_i.

8.3 Projection Space Example

Figure 8.3 shows a version of grammar G' with noise and missing data and Fig. 8.4 shows meta-operator additions to G'. In the problem instance defined by these grammar rules, $\mathcal{S}$ is the set of raw input data from sensors and Ω is defined by the rules of G' shown in Fig. 8.3. There is a single projection space, ω_1, and it is defined by $OP_{(\Omega,\omega_1)} = \{op_{32}, op_{33}, op_{34}, op_{35}\}$ and $OP_{(\omega_1,\omega_1)} = \{op_{30}, op_{31}\}$. The operators corresponding to rules 32, 33, 34, and 35 of G' project the base space to an abstract space and rules 30 and 31 map states in the projection space to other states in the projection space. The set $\Phi = \{op_{28}, op_{29}\}$. The operators corresponding

a.

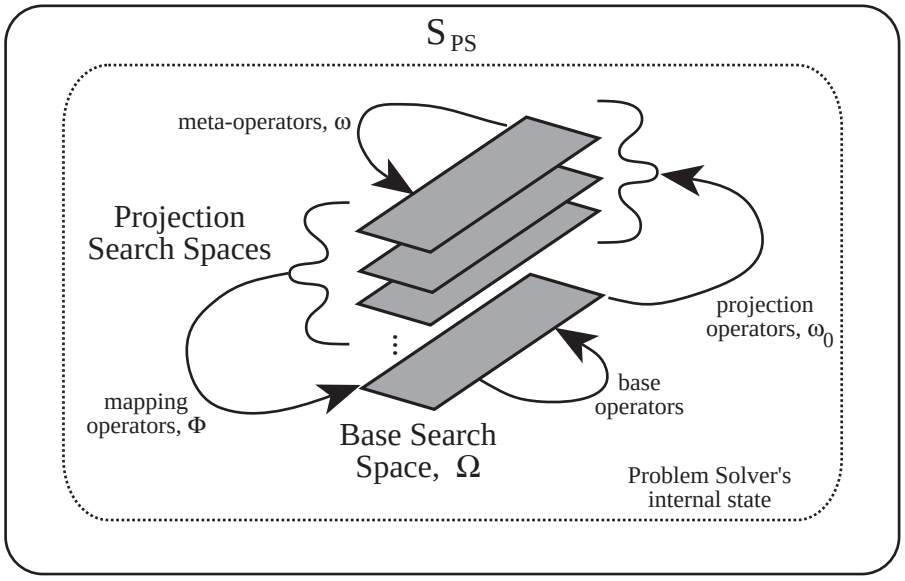


Extended State representation: operators applicable to each state are represented with the corresponding *Utility, Probability,* and *Cost* vectors.

Each vector entry consists of a measure of the *expected value* and a measure of the *variance.*

Expected values and variances are determined from IDP's cost and credibility functions. (In interpretation problems, credibility = utility.)

b.



UPC Summary

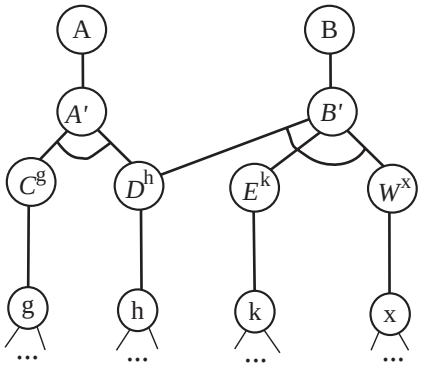
Figure 8.2. Overview of Extensions to the *UPC* Formalism

1. A → CD	7. f → (signal data)	16. M → Y
2. B → DEW	8. j → (signal data)	17.0 Y → qr
3.0 C → fg	9. g → (signal data)	17.1 Y → qhri
3.1. C → fgq	10. k → (signal data)	18. N → Z
4. E → jk	11. h → (signal data)	19. Z → xy
5.0 D → hi	12. x → (signal data)	20. O → X
5.1. D → rhi	13. i → (signal data)	21.1. X → fgh
6.0 W → xyz	14. y → (signal data)	21.2. X → fg
6.1. W → xy	15. z → (signal data)	

Interpretation Grammar G' with noise and missing data rules

Figure 8.3. G' Noise and Missing Data Rules

28. A→ A'
29. B→ B'
30. $A' \rightarrow C^{g}D^{h}$
31. $B' \rightarrow D^{h}E^{k}W^{x}$
32. $C^{g} \rightarrow$ g
33. $D^{h} \rightarrow$ h
34. $E^{k} \rightarrow$ k
35. $W^{x} \rightarrow$ x



(Signal Data)

Figure 8.4. Meta-Operators for Grammar G'

to rules 28 and 29 map the results of problem solving in the projection space back to the base space.

The *UPC* values for abstract states are calculated in the same manner as the calculation of *UPC* values for base space states. Figure 8.4 shows meta-operator additions to the grammar G' that will be used in an example to illustrate this. *UPC* values for abstract state D^h are shown in Fig. 8.5.

The conditional probability of A given D^h is, from Definition 6.3.13,

$$P(A \mid D^h) = \frac{P(A \cap D^h)}{P(D^h)} = \tag{8.1}$$

$$\frac{P(A \rightarrow D^h) + P(O \rightarrow D^h)}{0.6} = \tag{8.2}$$

$u_{D^h,30} = (1_A)$	$p_{D^h,30} = (0.5_A)$	$c_{D^h,30} = ((9,1)_A)$
$u_{D^h,31} = (1_B)$	$p_{D^h,31} = (0.33_B)$	$c_{D^h,31} = ((13,1)_B)$

Figure 8.5. *UPC* Vectors for Abstract State D

$$\frac{0.3}{0.6} = 0.5. \tag{8.3}$$

A is ambiguous with O, so

$$P(A \cap D^h) = P(A \rightarrow D^h + P(O \rightarrow D^h)). \tag{8.4}$$

$$P(A \rightarrow D^h) = P(A) * 1 = 0.2. \tag{8.5}$$

$$P(O \rightarrow D^h) = P(O) * 0.5 = 0.1. \tag{8.6}$$

Furthermore, $P(D^h) = P(h) = 0.6$. This is from the observation that the only element on the RHS of D^h is h, so

$$P(D^h \mid h) = 1 \Rightarrow P(D^h) = P(h). \tag{8.7}$$

The expected cost of a path to A' from D^h is 3. This assumes that each of the operators shown in Fig. 8.4 has cost 1. The expected cost of a path from A' to A is the expected cost of a generating an A minus the expected costs of g and h, or 6. Thus,

$$c_{D^h,30}(1,1) = 9. \tag{8.8}$$

The expected cost of a failed attempt to generate an A is always 1. When the problem solver attempts to extend D^h, it first tries to generate a path to A' including a g. The cost of failing to generate a g is 1, and, when g fails, the problem solver immediately ceases its attempt to generate a path from D^h to A.

The expected cost of attempting to generate a path from D^h to A is, from Definition 6.3.16,

$$p_{D^h,30} * 9 + (1 - p_{D^h,30}) * 1 = 0.5 * 9 + 0.5 = 5. \tag{8.9}$$

The *UPC* values for a path from D^h to B are calculated in a similar manner and are shown in Fig. 8.5. Given these values, the expected cost of attempting to generate a path from D^h to B is

$$p_{D^h,31} * 13 + (1 - p_{D^h,31}) * 1 = 0.33 * 13 + 0.67 * 1 = 4.96. \tag{8.10}$$

It is important to note that the addition of meta-operators also changes the *UPC* vectors for state h. (There are no meta-operators defined that are applicable to state f, so its *UPC* values do not change.) These changes, shown in Fig. 8.6, reflect the paths from h through the abstract states $A^{'}$ and $B^{'}$ to final states A and B. However, these additions to h's *UPC* vectors do not increase the cost of connecting h. This is because the cost of connecting a base space state is defined in terms of the costs of generating base space paths. Therefore, the addition of potential paths in projection spaces are not included in the calculation for the cost of connecting a state.

It is also important to note that the addition of potential paths in projection spaces can *reduce* the cost of connecting a state. This is a very important point and it will be discussed in more detail in the next section.

$u_{h,5} = (1_A, 1_B)$	$p_{h,5} = (0.33_A, 0.33_B)$	$c_{h,5} = ((7,3)_A, (10,3)_B)$
$u_{h,17} = (1_M)$	$p_{h,17} = (0.25_M)$	$c_{h,17} = ((5,1)_M)$
$u_{h,21} = (1_O)$	$p_{h,21} = (0.5_O)$	$c_{h,21} = ((4,1)_O)$
$u_{h,33} = (1_A, 1_B)$	$p_{h,33} = (0.5_A, 0.33_B)$	$c_{h,33} = ((10,2)_A(14,2)_B)$

Figure 8.6. *UPC* Vectors for State h Given Meta-Operator Extensions

8.4 Chapter Summary

The IDP/*UPC* framework extends the traditional notion of a search space to incorporate abstract and approximate states, and the operators that create, modify, and exploit them, in a unified representation including traditional forms of search-based problem solving. This chapter introduces and defines the concepts of *projection spaces* and *projecting* and *mapping* operators. Projection spaces are abstractions of a base search space in which an approximate, (hopefully) less costly version of a problem can be solved. Projection spaces are defined by special meta-level, projecting operators, and the results of problem solving in these spaces are propagated back to the base space by mapping operators. By defining projection spaces and the associated projecting operators, approximate/abstract operators, and mapping operators in terms of the as extensions of the base space, an integrated perspective of both domain and meta-level processing. This supports the analysis of problem solvers that use sophisticated control mechanisms to function incrementally and simultaneously in a continuum of abstraction spaces.

CHAPTER 9

POTENTIAL - THE BASIS FOR SOPHISTICATED CONTROL

As described in Chapter 6.4, an implicit (or in some cases explicit) objective of every interpretation problem solver is optimal processing. However, based on the definition of interpretation problem solving from Chapter 3, it might seem as if there is little an interpretation problem solver can do to enhance its efficiency. The reason for this is that interpretation problem solvers must, by definition, explore *every possible solution path* (i.e., connect the base space) in order to determine a solution. Recall that in Chapter 3 interpretation problems were defined as discrete optimization problems where a problem solver has to identify the "best" element of a set of interpretations, S. It is not sufficient for the problem solver to determine a single interpretation; the problem solver must consider *all* possible solutions and determine which is the "best." Given this requirement, the costs of problem solving appear to be dependent solely on characteristics of the domain such as its size, the cost of operator applications, etc., and are beyond the influence of the problem solver.

Fortunately, for many domains, control architectures can be formulated that do not require exhaustive enumeration of every search path in the base space and that enable an interpretation problem solver to connect every state in Ω very efficiently. These architectures use implicit enumeration actions that prune paths based on the structure of the search space without fully extending the paths. The IDP/*UPC* framework supports the analysis of a class of implicit enumeration strategies for interpretation problems that we defined as sophisticated control architectures. In Chapter 4, we demonstrated this analysis for a simple pruning algorithm.

In Chapter 6.4, Equations 6.4.4 and 6.4.1 defined the basis for making optimal control decisions from a local perspective. These equations specify that, from a local perspective, optimal processing is achieved when the operator chosen for execution maximizes the ratio $c(op_i)/cost(op_i)$ where $c(op_i)$ is the degree to which op_i reduces the expected cost of connecting all open states, and $cost(op_i)$ is the cost of executing op_i.

As discussed in Chapter 6.4.2, in many cases, the locally optimal control decision will not result in globally optimal problem solving. This occurs in situations where an operator on a search path does more than simply expand a state and extend a search path – where the operator actually *increases the information available to a problem solver regarding the interrelationships between partial solutions.* i.e., the operator increases the understanding of a partial solution's global significance. In these situations, the operator does not have to actually extend a search path in order to move the problem solver closer to termination, rather, the operator may alter the search space in some way that reduces the cost of problem solving (or increases the effectiveness of problem solving efforts) for some other set of operators. We will refer to this property of an operator as its *potential*.

The example used to demonstrate the problems associated with locally optimal control decisions is reproduced in Fig. 9.1. In this example, op_y alters the search space in some way to make it less costly to execute a series of operators that generate final state F. In addition, the dotted lines extending from F in Fig. 9.1 indicate operators that map the results of generating

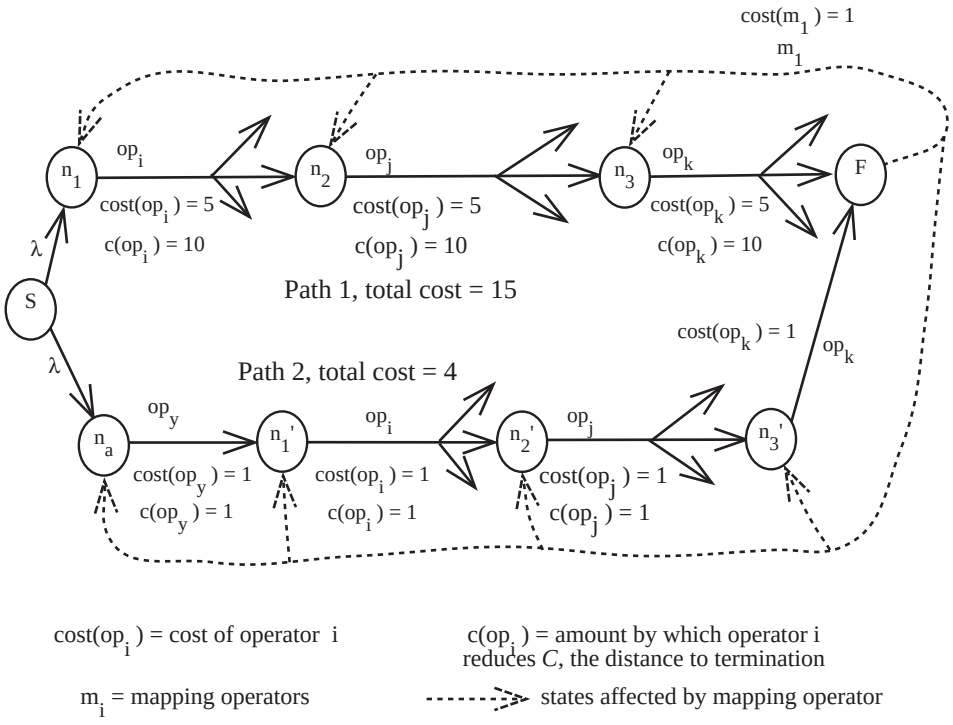


Figure 9.1. Example of the Non-local Effects of an Operator Application

F back to other search states and reduce the subsequent cost of problem solving. For example, the mapping operations may eliminate redundant activities. This could be implemented by mapping operators that are applied to state F that eliminate from consideration operators that generate paths solely to final states that are identical to F. Given these mapping operators, it now becomes feasible to overcome problems associated with locally optimal control decisions by taking into consideration the long-term implications of an operator. In this specific instance, op_y is superior to op_i because the long-term effects of its execution will be to generate final state F at a lower-cost and this will lead to the execution of a mapping operator that will prune redundant paths, including the path that begins with op_i. In such a situation, a problem solver must have some way of quantifying the effects of applying op_y that reflects the long-term benefits, i.e., the potential, of the operator.

The concept of potential is a critical element of the IDP/*UPC* analysis framework. It takes into account the changes in the *UPC* representation of base space states that occur as a result of the added information provided by an operator or a sequence of operator applications. Thus, potential is a mechanism that allows us to understand the interrelationships that exist between the current set of states (i.e., the search paths that have been created so far) and the states that can be derived from them. This includes an understanding of the long-term effects of an action. Potential relates to operations at all levels of processing, i.e., base space and projection space, but it is of particular importance to meta-operators that use abstractions or approximations. This is because potential can be used to address the question of how to evaluate the contribution made by meta-operators in terms that are consistent with the evaluation of problem solving actions that directly connect states in the base space. In general, meta-operators are not associated with

effects that can be quantified in the same way as the effects of base space operators. The effects of meta-operators are related more to long-term reductions in problem solving cost or increases in solution quality. Although a meta-operator may have no immediate effect on any base space search paths, which might appear to make it an undesirable choice of action, it may have a very significant long-term effect that reduces the expected cost of problem solving dramatically, making it a very good choice of action. In contrast, base space operators can be thought of as explicit enumeration mechanisms and they are associated with immediate effects resulting in the extension of base space search paths that can be easily quantified.

Formally,

Definition 9.0.1 *Potential of operator* op_i *applied to state* s_n, $Pot(op_i, s_n) = FTN_{\forall S'}\ (P(S' \mid s_n), Potential(S'), cost_g(S', s_n), cost_m(S'))$, *where* FTN *is a function that is described below, each element* S' *is a state that can be reached from* s_n *that increases the information available to a problem solver regarding the interrelationships between partial solutions (i.e., a state with potential),* $P(S' \mid s_n)$ *is the probability of generating* S' *given* s_n, $Potential(S')$ *is a measure of the degree to which* S' *reduces the cost of problem solving,* $cost_g(S', s_n)$ *is the expected cost of generating* S' *given* s_n, *and* $cost_m(S')$ *is the cost of realizing* $Potential(S')$, *i.e., the cost of mapping* S' *back to the base space.*

It is important to point out that this definition of potential is based solely on the characteristics of s_n and the statistical properties of the domain calculated from the IDP definition. It does not take into account the existence, or absence, of any other states. In order to account for other states, you have to create abstract states in a projection space that represent the relationship between the states. Thus, all non-local implications must be calculated using meta-level actions to find relationships between base level states. Once this is accomplished, this equation will take any state relationships into account.

$FTN_{\forall S'}$, the general computation of $Pot(op_i, s_n)$, is $(P(S' \mid s_n) * (Potential(S') - cost_g(S', s_n) - cost_m(S')))$. This is for situations where the cardinality of S' is 1. For situations where an operator, op_i, represents the first step on paths to multiple S'_i, the computation is more complicated. In these situations, the function FTN determines $Pot(op_i, s_n)$ based on the relationships between the states S'_i. These relationships are defined in terms of the paths from s_n to the states S'_i (i.e., the costs $cost_g(S', s_n)$) and the set of states that are affected when the potential of the states S'_i is mapped back to the base space. Figure 9.2 depicts the possible relationships between states S'_i in terms of search space paths and base space states.

Intuitively, it is easy to think of the benefits of potential as being cumulative. If this were true, the distance to termination should be reduced by the sum of the potential, $Potential(S'_i)$, of all states S'_i. However, Fig. 9.2 shows a situation where this is not correct. Specifically, when the sets of states affected by the mapping functions that propagate the potential of the states, S'_i, back to the base space interact, some of the benefits of the potential might overlap or be redundant. In this case, summing the potential of the states, S'_i, would give an overestimate of the expected benefits.

Similarly, it is easy to develop an intuitive perspective of the costs of generating the states, S'_i, as a sum. However, Fig. 9.2 shows situations where this is not correct. When the paths to the states, S'_i, interact or overlap, summing results in an overestimate of the costs and an underestimate of the benefits of the potential associated with the states, S'_i.

The general formulas for calculating the expected costs of reaching states with potential, S_i, and the effects of mapping the potential back to the base space, are:

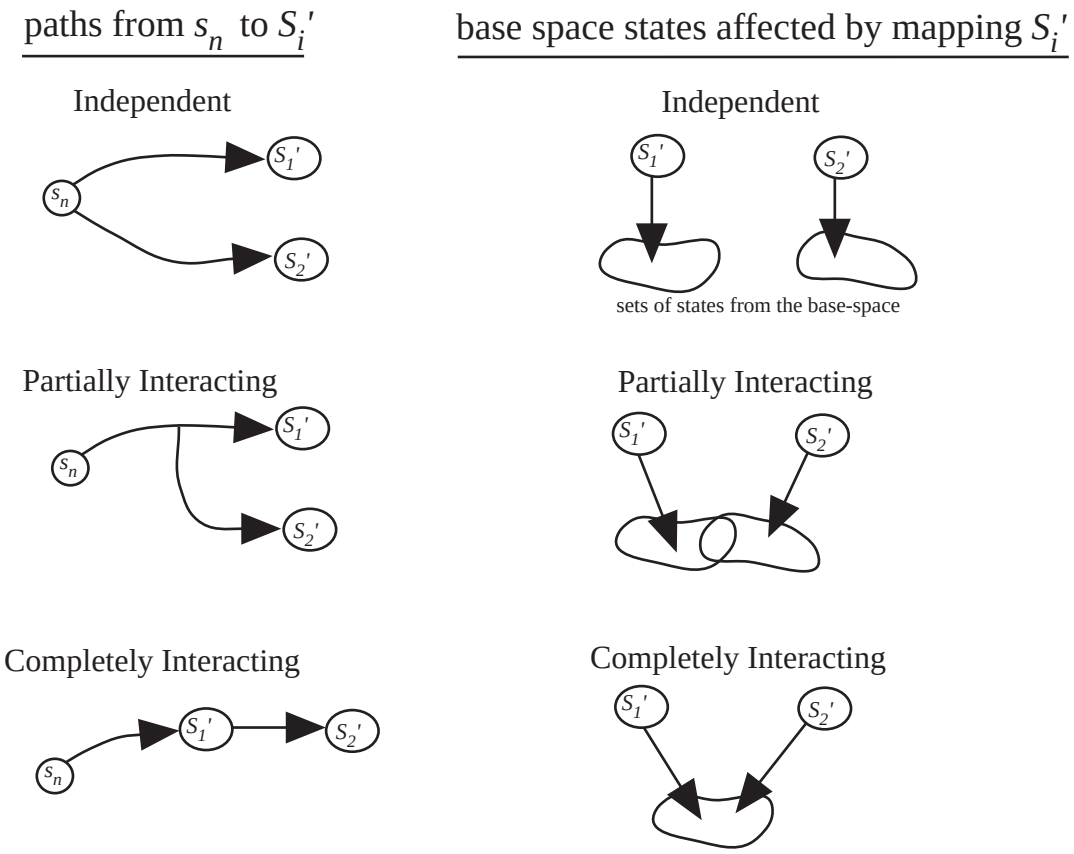


Figure 9.2. Relationships Between States with Potential

Definition 9.0.2 *Expected cost of generating states,* S_i = $cost_g(S_1', s_n)$ + $cost_g(S_2', s_n)$ + . . . + $cost_g(S_m', s_n)$ - $cost_\cap(S_1', S_2', \ldots S_m', s_n)$, *where* m *is the number of states,* S_i, *and* $cost_\cap(S_i', S_j', \ldots, s_n)$ *is the intersection of the paths from* s_n *to the states* $S_i', S_j', \ldots$. *In other words, the expected costs are summed, then all possible interactions are computed and subtracted from this cost. In situations where the states,* S_i *are independent, the intersection terms are all 0 and this simplifies to the sum of the costs. In situations where the paths to the states are completely interacting, this simplifies to the maximum cost path.*

Definition 9.0.3 *Expected benefits of mapping potential from states* S_i *back to base space =* $Potential(S_1')$ + $Potential(S_2')$ + . . . + $Potential(S_m')$ - $potential_\cap(S_1', S_2', \ldots S_m')$, *where* m *is the number of states,* S_i, *and* $potential_\cap(S_i', S_j', \ldots)$ *is the intersection of the expected potential derived from mapping the states* $S_i', S_j', \ldots$ *back to the base space. In other words, the expected benefits of the potential of the states,* S_i' *are summed, then all possible interactions are computed and subtracted from this cost. In situations where the states,* S_i *are independent, this is the sum of the potentials. In situations where the sets of base space states are completely interacting, this is the maximum of the* $Potential(S_i')$.

In general, the costs $cost_m(S_i')$ of mapping the potential of states back to the base space must be treated in a manner similar to that discussed above.

In the experiments in the following chapters, the paths to states with potential are all independent, so their costs are summed, and the sets of states affected by mapping functions are all completely interacting, so the maximum potential is used.

The following definition will be used in discussions of potential.

Definition 9.0.4 *Distance to Termination,* C *- The expected amount of processing an interpretation problem solver must perform before an answer is determined. More formally,* C *is the sum of the expected costs of connecting all open states in* Ω.

For operator op_i, $Pot(op_i, s_n)$ represents the degree to which it reduces C by altering the nature of a search space S_{PS} exclusive of extending any search paths. We will give an example of calculating $Pot(op_i, s_n)$ in Chapter 9.3. $cost_m(S')$ represents the cost of mapping operators that would be used to realize the potential of an abstract state in a projection space. This cost is also associated with actions in the base space, such as bounding operators, that are also used to realize $Pot(op_i, s_n)$.

Given the definitions of $Pot(op_i, s_n)$ and C, all potential operators can be judged based on their expected impact on the distance to termination. Each potential operator will extend one or more paths[1] by an expected amount and the "value" of an operator can be determined by the degree to which this amount reduces the distance to termination. Figure 9.3 illustrates the computation of distance to termination.

A problem solver's distance to termination can be determined by summing, for each intermediate state not fully expanded, the expected cost of extending paths to all potential final states that can be reached from the intermediate state. In addition, the computation of distance to termination must factor in some notion of the *potential* of available courses of action.

The potential associated with a path is based on the subsequent availability of a mapping operator capable of recognizing and exploiting the emerging structure of the problem instance. For example, by modifying the *UPC* values of the base space. (An example will be presented in the next section and a taxonomy of mapping operators is defined in Appendix B.) Intuitively, the process of recognizing an emerging problem structure is analogous to viewing the search space from a high vantage point where it is possible to discern general features of the problem space that can be used to guide and improve the efficiency of problem solving efforts. In effect, this extends the representation of a search space to a three dimensional space where the topology of a search space is defined by potential.

The relationship between potential and the distance to termination is illustrated in Fig. 9.4. In this figure, C represents the expected cost of problem solving, i.e., the distance to termination. C' represents the cost of problem solving in a new search space, one created by the application of operator op_i. If op_i does not alter the properties of the search space S_{PS} in any appreciable way, then $C = C' + cost(op_i)$. In other words, the only difference between the two search spaces S_{PS} and S'_{PS} (which is the result of applying op_i to S_{PS} is that paths associated with op_i have been extended. If op_i does alter the properties of the search space S_{PS} in a way such that $C > C' + cost(op_i)$, then the difference is what we refer to as potential.

As an example, consider a domain with bounding functions that are not based on a predetermined threshold, but that are based on the results of problem solving. If, for example, the domain uses the credibility of any full interpretations it derives to prune paths during

[1] In more complex grammars, for example, those that include *noise* (which is defined in Chapter 4.1.2), it is possible for a single operator execution to extend multiple paths from a state.

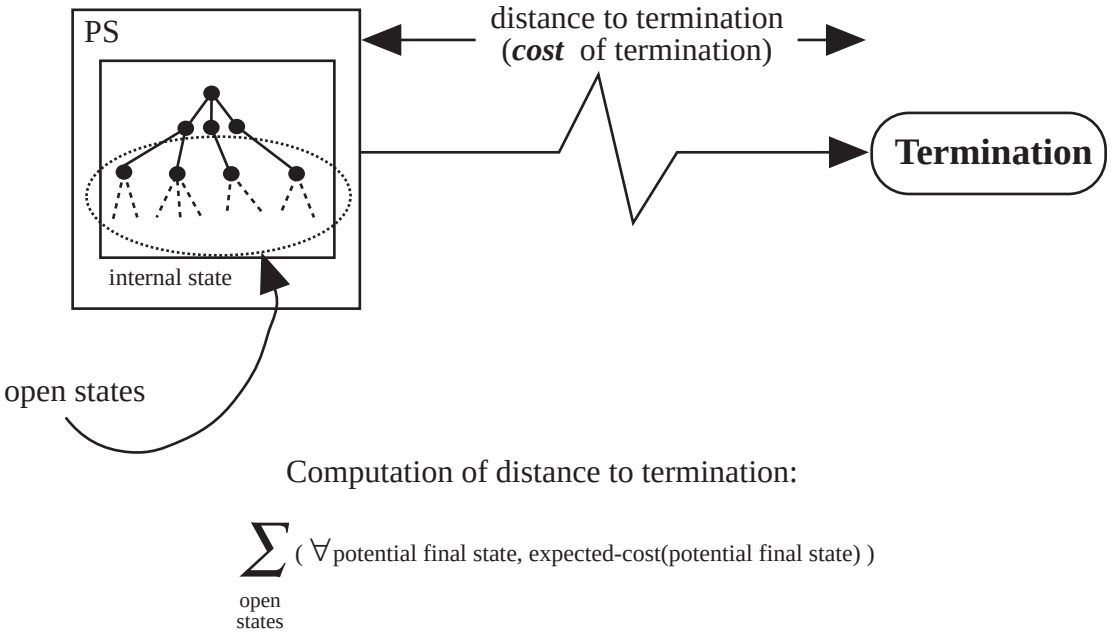


Figure 9.3. Representation of a Problem Solver's Distance to Termination

subsequent search, the creation of a full interpretation alters the nature of the search space in a significant way. In essence, the creation of a full interpretation that is used to prune other potential search operations transforms the existing search space, S_{PS}, into a new search space, S'_{PS} that is (hopefully!) less costly to search.

9.1 Calculating Potential

As discussed in the previous section, the potential of a state, $Potential(S')$, is based on the degree to which the existence of S' reduces the expected cost of problem solving. This can be formalized as:

Definition 9.1.1 *Potential of a state,* $S = Potential(S') = C_{s_n,t} - C_{S',t',s_n,t}$, *where* $C_{s_n,t}$ *represents the expected cost of problem solving given the existence of state* s_n *at time* t, $C_{S',t',s_n,t}$ *represents the expected cost of problem solving given the existence of state* s_n *at time* t *and state* S' *at time* t' *and where* $t < t'$.

This definition distinguishes between $C_{s_n,t}$ and $E(C)$. This distinction is necessary because the existence of s_n may imply a great deal about the nature of a search space. For example, consider the situation shown in Fig. 9.5. The figure illustrates two different sets of interpretation trees, one which includes the terminals x_i and one which includes the terminals y_i. The interpretations that include an x_i do not include a y_i, and vice-versa. Assume that a given problem instance is equally likely to involve an interpretation from either set and that the expected cost of generating interpretations for problem instances from the X set is much less than the Y set. Assuming that there is no overlap between the two sets of interpretations, $E(C)$ will be an average of the expected costs of interpretations of each of the sets. This average

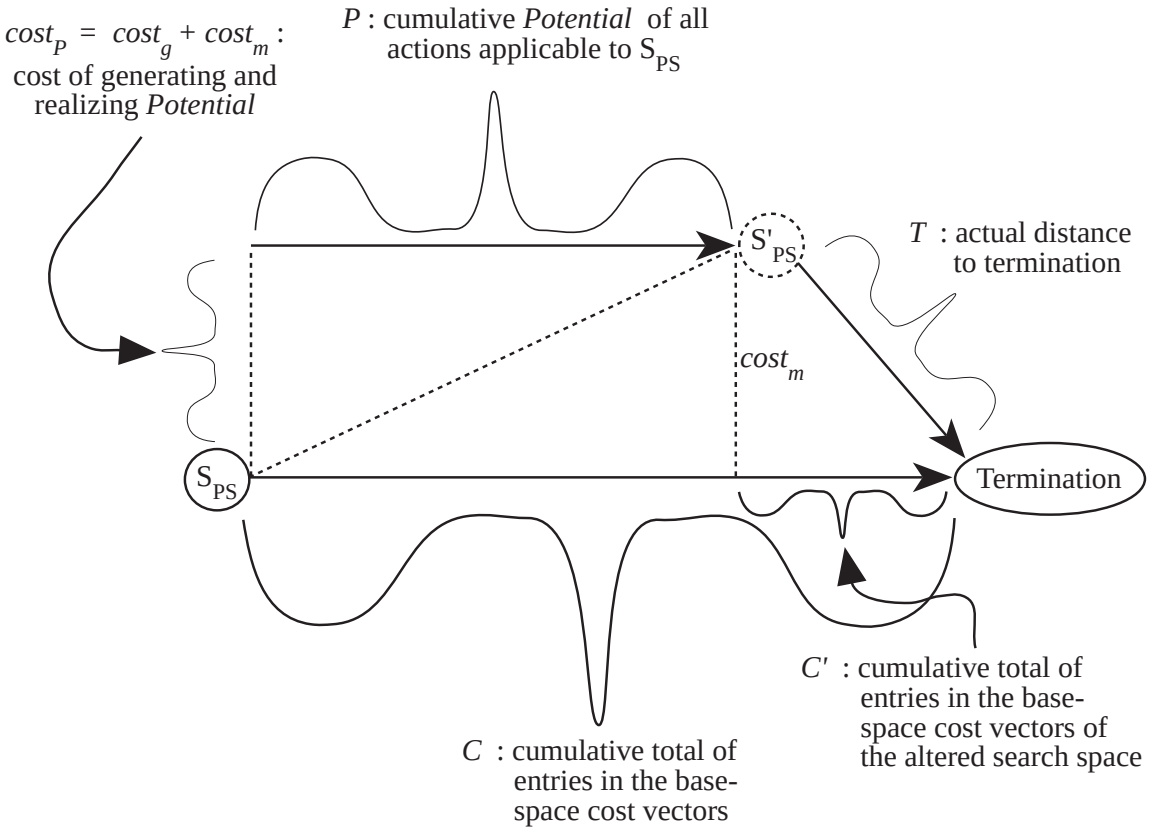


Figure 9.4. A Basic Representation of Potential and Distance to Termination

will be much larger than $E(C)$ for interpretations from the X set and it will be much smaller than $E(C)$ interpretations from the Y set.

Now consider what happens when a specific state, x_j is generated. The problem solver would be able to determine that the expected distance to termination should be based on the average cost of interpretations for the set of X interpretations, not on $E(C)$. Likewise, if a specific state y_k is generated, the problem solver would be able to determine that the expected distance to termination should be based on the average cost of interpretations for the set of Y interpretations. Consequently, it should be clear that the exact computation of $Potential(S')$ must consider the implications the existence of the local state, s_n, has on the expected distance to termination. The exact computation must similarly consider the implications S' has on the expected distance to termination.

Furthermore, the time, t at which a state, s_n is created and the time, t', at which a state, S' is expected to be created must also factor into the computation of $Potential(S')$. If t is relatively large, indicating that the distance to termination is short, the expected benefits associated with the existence of S' may be minor. For example, all the paths that may have been pruned given the existence of S' may have already been generated.

On the other hand, if t is relatively small, then the benefits associated with the existence of S' may be more significant. This is an especially relevant point in domains where a state's potential is a function of its credibility. The early existence of s_n may indicate that it has an

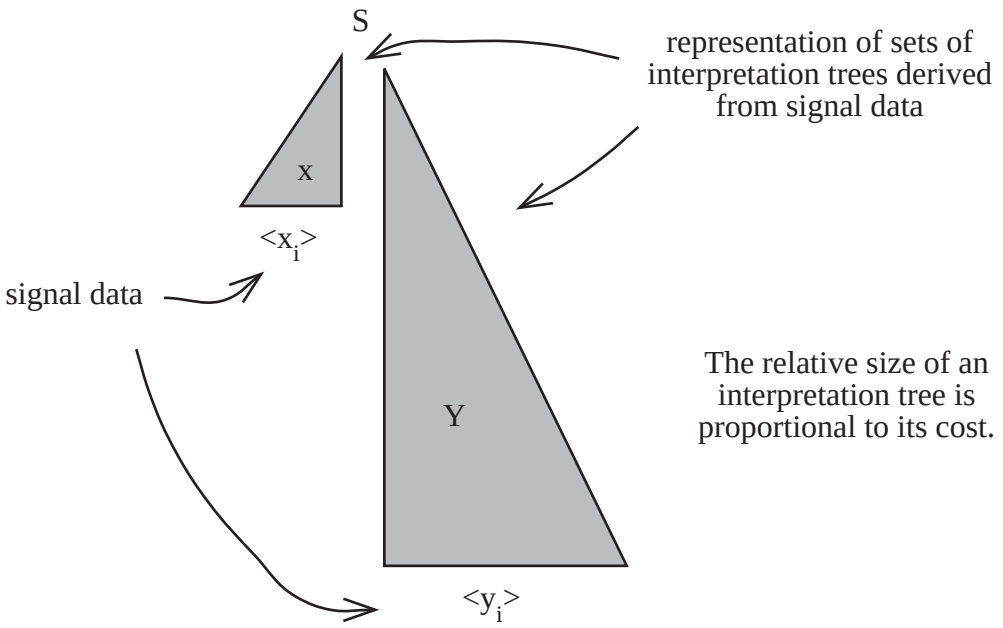


Figure 9.5. Implied Information Associated with a State

exceptionally high credibility and, if there is a strong correlation between the credibility of s_n and the credibility of S', S' may have an exceptionally high potential.

It may also be necessary to consider the characteristics of a state directly. For example, rather than making inferences about the nature of a search space based on implied information about the relative value of a credibility, it may be necessary to use a direct consideration. This may also extend to other properties of a state, depending on a domain.

We have extended the algorithms presented in Chapter 5 so that they can be used to calculate $C_{s_n,t}$ and $C_{S',t',s_n,t}$. The extensions involve using s_n and S' to map the grammar, IDP$_G$, used to calculate $E(C)$, to a new grammar IDP$_{G'}$ that is used to calculate $C_{s_n,t}$ or $C_{S',t',s_n,t}$. Similarly, the values of t and t' (and other relevant characteristics of a state, such as credibility) are used to map IDP$_{G'}$ to another new grammar, IDP$_{G''}$. These mappings do not involve the creation of entirely new grammars. Rather, they primarily involve adjusting the distribution functions in the existing grammar.

The first mapping and expected distance to termination calculation is accomplished as follows:

- For state s_n, determine the likelihood of reaching all the potential final states (SNTs) on paths from s_n.
- Normalize these values.
- Replace the distribution function ψ_S, which specifies the distribution of the SNTs in the grammar, with the normalized values. For SNTs that are not on paths from s_n, set their distributions to 0.
- Replace the distribution function ψ_{s_n}, which specifies the distribution of subtrees generated from s_n, with 0. This indicates that the state, s_n, has already been generated

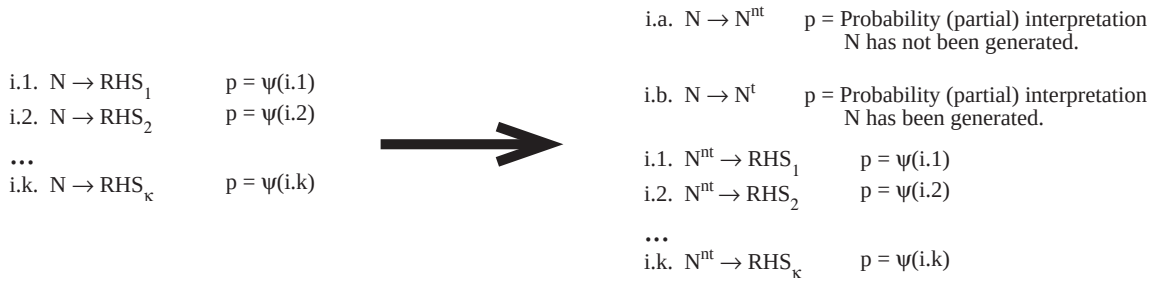


Figure 9.6. Grammar Transformation for Calculating Potential

and the associated cost should be excluded from further estimations of distance to termination. In effect, this makes s_n a terminal symbol (if it is not already a terminal symbol).

- Calculate C_{s_n} using the algorithm for calculating $E(C)$ from Chapter 5.4.

The second mapping and expected distance to termination calculation is a little more complex. It requires that a fundamental transformation be made to the original grammar, IDP$_G$. The new grammar will be represented as IDP$_{G''}$ The transformation is shown in Fig. 9.6. In effect, for each nonterminal of the grammar, N, a new rule is added with the form $N \rightarrow N^{nt} \mid N^t$. In the new rule, N^{nt} represents a nonterminal and N^t represents a terminal symbol. In all other rules of the grammar that include N on the left-hand-side of the rule, N is replaced with N^{nt}. (Note that there should not be any recursive rules, so N will not be on both the left and right hand side of the same rule.) The distribution function, ψ_N, specifies whether an N should be interpreted as a nonterminal (N^{nt}), in which case it can be used to generate additional interpretation subtrees, or whether an N should be interpreted as a terminal symbol, N^t. When N is interpreted as a terminal symbol, it means that the state already exists and the costs associated with deriving it should be ignored.

To calculate $E(C)$, the distribution function for N will always generate N^{nt}. This will result in a grammar that produces problem instances that are identical to the original grammar, IDP$_G$. After some time, t, the current state of problem solving is modeled by adjusting the distribution function for N. The adjustment is a function of time, t, and other factors including the characteristics of a state such as credibility. The adjustment will increase the probability that an N generates an N^{nt}. This corresponds to the probability that the partial interpretation corresponding to N has been generated at time, t. Now, to calculate the expected distance to termination, $C_{S',t',s_n,t}$ the following procedure is used:

- For each nonterminal element, N, of the grammar, adjust the distribution function ψ_N, which specifies whether or not a partial interpretation corresponding to N has been generated. The adjustment is made based on a function of time, t, and other relevant factors such as the credibility of the state s_n, etc.

- Calculate $C_{S',t',s_n,t}$ using the algorithm for calculating $E(C)$ from Chapter 5.4. (Note that the calculation of $C_{s_n,t}$ must also use this technique.)

Interpretation Grammar *G'*

0. S → A | B	2. B → DEW
1. A → CD	4. E → jk
3. C → fg	6. W → xyz
5. D → hi	8. j → (signal data)
7. f → (signal data)	10. k → (signal data)
9. g → (signal data)	12. x → (signal data)
11. h → (signal data)	14. y → (signal data)
13. i → (signal data)	15. z → (signal data)

Figure 9.7. Interpretation Search Operators Shown as a Set of Production Rules

9.2 An Example of Potential

Consider the implications associated with the structure of a complex search space. In particular, consider the fundamental structure of a convergent search space. In these domains, the measures relating a state to the potential final states it implies are not dependent solely on the characteristics of the state. They also depend to a large degree on the existence, and characteristics, of other intermediate states in the search space. As a consequence, the calculation of the measures must be made dynamically.

This will be illustrated with a simple example using the problem solving grammar introduced in Chapter 3. The rules of this grammar are shown again in Fig. 9.7.

In this grammar, A and B are the SNTs, or solution-nonterminals, that define the potential final states in the search space. For this example, assume that S's distribution functions are both 0.5. Thus, the start symbol generates an A with probability 0.5 and a B with probability 0.5.

Figure 9.8 represents the *UPC* values for paths from a state, h, of the convergent search space derived from this structure. The two potential final states associated with h are A and B. Some of the necessary details to compute full *UPC* values for the paths from h to A and from h to B have been omitted, and the values for cost and utility are shown as functions as a result. However, in this example we will concentrate on the values for the expected probability of successfully reaching a potential final state. It should be clear that, given an h, the probability of generating an A is the same as the probability of generating a B. Both are 0.5.

Though this is a relatively trivial example, it is, by definition, a complex domain since the search paths interact. Specifically, consider the case of the problem solver's internal database containing both an h and an f. These states interact in that they only coexist in one situation, when the complete interpretation is an A. This is because, by definition, an interpretation must explain all the data and, in this example grammar, an A and a B cannot be generated in the same derivation. Therefore, the only interpretation that includes both an h and an f is A. This observation is represented in Fig. 9.8. Here, an abstract state called f, h is shown along with the associated *UPC* values.

Now consider the implications of this observation, some of which are illustrated in Fig. 9.9. In this figure, we assume that the expected cost of generating an f or an h is 1, the expected cost of generating an A is 8, and the expected cost of generating a B is 11. For this example, we will ignore credibility calculations.

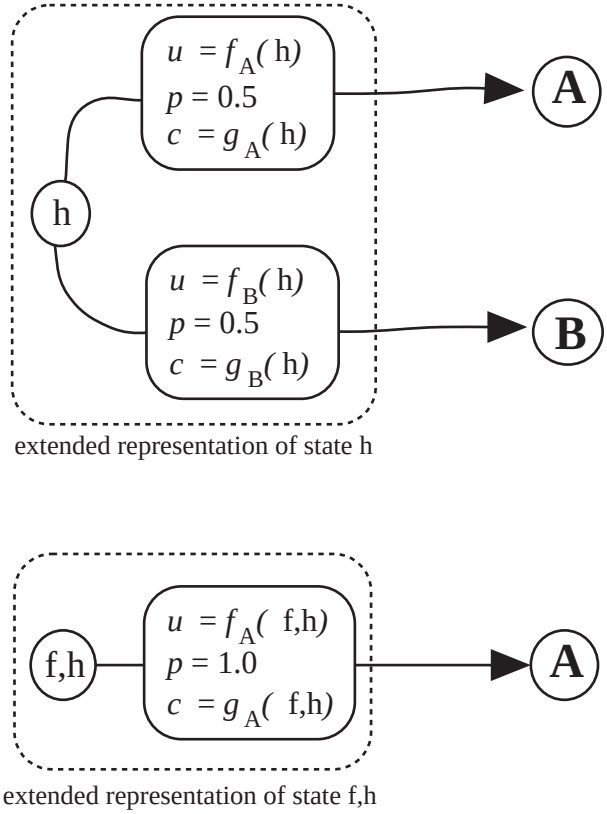
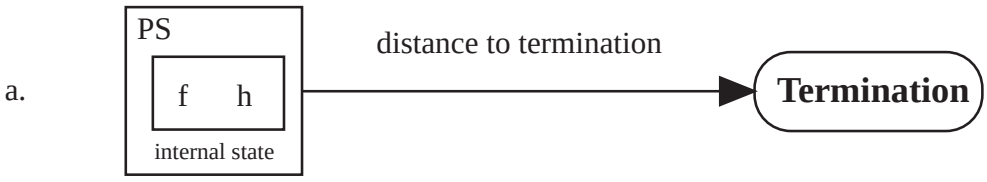


Figure 9.8. Representation of the *UPC* Values for Base- and Abstract States



Computation of distance to termination:

open state: f ⇒ potential final state A

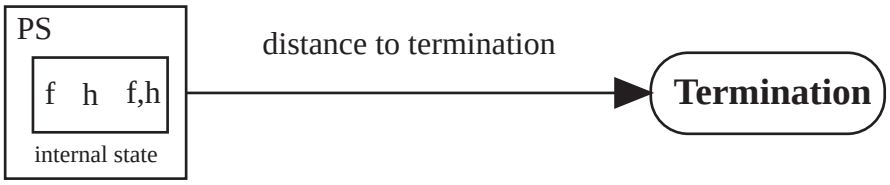
cost associated with open state f =
P(A | f)*(cost(A) - Cost(f)) = 1*(8 - 1) = 7

open state: h ⇒ potential final states A, B

cost associated with open state h =
P(A | h)*(cost(A) - Cost(h)) = 0.5*(8 - 1) = 3.5 +
P(B | h)*(cost(B) - Cost(h)) = 0.5*(11 - 1) = 5

Total distance to termination = 7 + 3.5 + 5 = 15.5

Figure 9.9. Calculating Distance to Termination



Computation of distance to termination
(after state f,h has been generated and
mapped back to the base-space):

connected state: f ⇒ nil

cost associated with
connected state f = 0

open state: h ⇒ potential final states A

cost associated with open state h =
P(A | f,h)*(cost(A) - Cost(h) - Cost(f)) =
1*(8 - 1-1) = 6

P(B | f,h)*(cost(B) - Cost(h) - Cost(f)) =
0*(11 - 1 -1) = 0

Total distance to termination = 6

Figure 9.10. Effects of Abstract Processing on Distance to Termination

In Fig. 9.9, the distance to termination is determined by summing the expected costs of the paths from states h and f to their associated potential final states. For this example, we will use the simplified calculation:

E(cost from s_n to F) = $P(F \mid s_n) * (E(cost_F) - cost_{s_n})$

i.e., the expected cost of a path from intermediate state s_n to a potential final state F is the probability that the final state can be reached from s_n multiplied by the expected cost of deriving the final state minus the cost of generating s_n. (Adjusting the expected cost of generating the final state is necessary because the cost of generating the intermediate state, s_n, is included in the expected cost of generating the final state, but this cost has already been incurred and will not recur as a path is extended from s_n to the final state. A full discussion of calculating *UPC* values accurately is given in Chapter 6.)

Now consider the implications of the existence of the abstract state f, h. Given this state, the problem solver would know that B could not be a solution (for the reasons discussed above) and it could adjust the *UPC* values for states f and h accordingly. The relationship represented by the state f, h is therefore an important one. To exploit this relationship, an operator would be added to the grammar that creates the state f, h and a mapping operator would also be added that would propagate the effects of f, h back to the base space. In this situation, the existence of states f and h would be associated with a new operator. However, the execution of this operator and the corresponding mapping function, are not related to the cost of termination in the sense that they directly reduce the cost of termination by extending paths in the base space and, as a consequence, the problem solver's control component has no way to evaluate their worth compared with alternative actions. Thus, the question becomes, how do we represent the value of the operator that generates the state f, h? This value is what we refer to as *potential*

and we will compute it as the expected reduction in cost associated with the existence of a state, or $P(state) * (\text{expected-cost-reduction})$.

The calculation of potential is illustrated in Fig. 9.10. In Fig. 9.10, the distance to termination is calculated based on the *UPC* values of states f and h after they have been modified to reflect the existence of state f, h. The difference between the distances to termination for the two examples is $15.5 - 6 = 9.5$ In other words, the expected cost to termination from the problem solving states in Fig. 9.9 is 9.5 units more than the expected cost to termination from the problem solving states in Fig. 9.10. If we assume that the cost of generating state f, h is 1 and the cost of mapping the implications of f, h is also 1, then the expected cost reduction associated with f, h is $9.5 - 1 - 1 = 7.5$. Furthermore, the probability that f, h can be generated given an h is 0.5, so, from state h, the potential of the operator that will generate state f, h is $0.5 * 7.5 = 3.75$.

It is also important to observe that abstractions and approximations are useless without operators designed explicitly to exploit them. In other words, it does no good to climb a tree for a better look at the surrounding territory if you have no way to get down from the tree. In the IDP/*UPC* framework, we refer to the functions that transfer the results of problem solving in an abstract space back to the base space as *mapping functions.* In Chapter 9.3, the mapping functions will be critical components for calculating the potential of an action. In other words, the potential of a given action will be based on the degree to which subsequent mapping functions can exploit the results of the action to modify the base space in a way that reduces the cost of problem solving.

9.3 Incorporating Potential in Control Decisions

Using Definition 9.0.1, we have altered the objective strategy used in the simulator to include a consideration of an action's potential as well as its direct effects on the distance to termination, C. Based on the definition for the calculation of potential from Chapter 9.1, we have developed mechanisms for calculating potential for two forms of processing, bounding functions and approximate processing operators. In this section, we will discuss the approximate calculation of the potential associated with bounding functions. This calculation is useful in situations where the cost of calculating precise values of potential are prohibitively high. This approximation is effective in situations where it maintains the relative ordering of problem solving activities.

As discussed previously, the paths to states with potential are all independent, so their costs are determined individually and summed where appropriate, and the sets of states affected by mapping potential back to the base space are completely interacting, so the maximum potential is used to determine $Pot(s_n, S')$ where appropriate. In the next section, we will present the preliminary results of experiments that incorporate this information in their problem solving decisions.

The original objective strategy was defined in Chapter 6.4.2 and it was based on choosing the operator that maximized the equation $\frac{c(op_i)}{cost(op_i)}$, where $c(op_i)$ is the degree to which op_i directly reduces C and $cost(op_i)$ is the cost of executing op_i. The evaluation function for operators used for this set of experiments was changed to $\frac{(c(op_i)+Pot(op_i,s_n))}{cost(op_i)}$, where $Pot(op_i, s_n)$ is the potential of operator op_i applied to state s_n.

The potential associated with bounding functions is based on the expected utility of the final states, F_i, that can be reached along paths including op_i. To calculate this potential, the

problem solver first calculates the cost of problem solving in a search space where the bounding functions have a cutoff threshold equal to the expected credibility of the potential final states, F_i, that can be reached along paths including op_i. This calculation is made for each of the potential final states and will be called C_{F_i}. For each of the final states, F_i, its potential, $Potential(F_i)$, can be approximated by $C - C_{F_i}$, where C is the current expected cost to termination. (Note that the exact value of $Potential(F_i)$ is given by $C_{s_n,t} - C_{F_i,s_n,t}$, where $C_{s_n,t}$ is the expected cost of problem solving given that state s_n has been generated at time t, $C_{F_i,t',s_n,t}$ is the expected cost of problem solving given that state s_n has been generated at time t and that state F_i has been generated at time t', and $t < t'$.) Next, the problem solver references the expected cost of reaching the final states from the *UPC* vectors. This is the cost of generating the potential. From Definition 9.0.1 this quantity is referred to as $cost_g(F_i, s_n)$. For the grammar used in these experiments, the bounding functions are "built in" – they are executed as part of the normal course of problem solving and are already factored into the calculation of C_{F_i}. Therefore, the cost of realizing the potential, $cost_m(F_i)$, is 0.

Given that a specific potential final state can be reached, the actual distance to termination is then $C_{F_i} + cost_g(F_i, s_n)$, and the *net potential* of the action is the probability that the specific potential final state can be reached multiplied by $Potential(F_i)$ minus the cost of reaching the final state, or $P(F_i \mid s_n) * (Potential(F_i) - cost_g)$. For a given operator, the maximum value is taken for all potential final states that can be reached along paths from s_n that include the operator. This value is used in the problem solver's operator rating function as $Pot(op_i, s_n)$.

To summarize this calculation, the following are the steps used to compute the approximate potential of an operator, op_i, applied to a state, s_n:

1. Determine C, the current expected cost to termination. $C = E(C)$– expected-cost(s_n).

2. For s_n, determine the expected utility of the potential final states, F_i, that can be reached along paths from s_n starting with op_i. This is done dynamically by referencing the *UPC* values of s_n. The expected utility of F_i appears in the utility vector corresponding to op_i. Each F_i corresponds to an S' in Definition 9.0.1.

3. For each of the potential final states, F_i, compute C_{F_i}; the expected cost of problem solving for an IDP with bounding functions that use a cutoff threshold equal to the expected credibility of F_i. $C_{F_i} = E(C_{F_i})$– expected-cost(F_i). The cost of problem solving given F_i is the cost of connecting the search space using a bounding function cutoff threshold equal to F_i's utility minus the cost of generating F_i, which already exists.

4. For each of the potential final states, F_i, compute $Potential(F_i)$ = C - C_{F_i}.

5. For each potential final state, F_i, that can be reached from s_n, the cost, $cost_g(F_i, s_n)$, of generating the associated $Potential(F_i)$ is the expected cost of a path from s_n to F_i. This value is referenced in the *UPC* values.

6. Finally, $Pot(op_i, s_n) = max_{\forall F_i} P(F_i \mid s_n) * (Potential(F_i) - cost_g(F_i))$.

9.4 Initial Experiments with Potential

We have implemented the process for calculating potential described above and we have used it in several experiments. These initial experiments were intended to explore the effects of

the use of potential. In these experiments, $E(C)$ was calculated using a pruning threshold that is a function of the expected credibility of the "correct" result of a problem solving instance and the equations and procedures defined in Chapter 5. In general, this will yield an approximation of the actual value of $E(C)$. However, in situations where the final state(s) used to specify a pruning threshold is (are) generated before any of the states that they eventually prune are created, this calculation will be exact. This is not an unreasonable assumption to make, even in non-monotonic domains. This is especially true if the pruning threshold is a function of the credibility of the final state(s) such as "pruning threshold = 0.5 * best credibility of the final states generated so far."

In these experiments, conditional probabilities and values for $\Omega(s)$ for all elements of the grammar are computed a priori. The problem solver computes expected credibilities, costs, and probabilities dynamically, using the previously computed values for conditional probability and Ω.

The most striking effect is in the increased time required to conduct an experiment. Given that the rating of every problem solving action requires costly computations, i.e., the computation of C_{F_i}, this result was not surprising. Furthermore, we have found that the cost of the computation of $C(S'_{PS})$ can be reduced significantly through the use of dynamic programming techniques.

Another significant result is that the use of Potential substantially reduces the cost of problem solving. This is consistent with our intuitions that bounding functions reduce the expected cost of problem solving, but the grammar used in these experiments was designed specifically to demonstrate the advantages of bounding functions and this result may not be consistent with real-world performance. (To demonstrate the advantages of bounding functions, potential interpretations derived using noise and missing data rules were given credibility ratings significantly lower than other paths. In addition, the distribution of noise and missing data in the generation of problem instances was less than 10%. In other words, most of the correct interpretations derived did not involve the use of noise or missing data rules.)

Table 9.1 summarizes experiments 18, 19, and 20; the experiments with potential and bounding functions. The experimental methods were identical to those used in the first set of experiments.

9.4.1 Experiments 1 and 18

In Experiment 18, the grammar used was identical to previously used versions of G' except that the bounding function used was based on dynamic thresholds. Experiment 1 is included for comparison. In 18, the problem solver ran faster with the bounding functions and it did not mistakenly prune any correct paths. Compared with Experiment 1, the performance improvement is noticeable. It is interesting to note that the use of dynamic bounding functions improved the percentage of correct answers generated when compared with Experiments 8 through 17.

9.4.2 Experiment 19

For this experiment, we extended the grammar G' by including new nonterminals that have RHSs with multiple occurrences of nonterminals from G'. This extension was intended to approximate a grammar representative of a vehicle tracking domain such as the domain

Table 9.1. Results of Verification Experiments – Potential

Exp	Generation				Interpretation				Sig	% C
	G	Dist	U	$E(C)$	G	Dist	U	Avg. C		
1	1	even	0.5	201	1	even	0.5	200	N	100
18	4	even	0.5	179	4	even	0.5	182	N	100
19	5	even	0.5	2,231	5	even	0.5	2,234	N	100
20	6	even	0.5	1,639	6	even	0.5	1,658	N	100

Abbreviations

Exp: Experiment

G: The problem solving grammar used;
1: G'
4: G' and dynamic bounding functions with cost 1
5: $G2$
6: $G2$ and dynamic bounding functions with cost 0.01

Dist: Distribution of Domain Events;
even: domain events evenly distributed

U; expected problem instance credibility;
0.5: problem instances have expected credibility 0.5

$E(C)$: Expected Cost of problem solving for given grammar

Avg. C: actual average cost for 100 samples of 50 random problem instances each

Sig: Whether or not the difference between expected cost and
the actual average cost was statistically significant
Y: yes, there is a statistically significant difference
N: no, there is not a statistically significant difference

% Correct: percentage of correct answers found

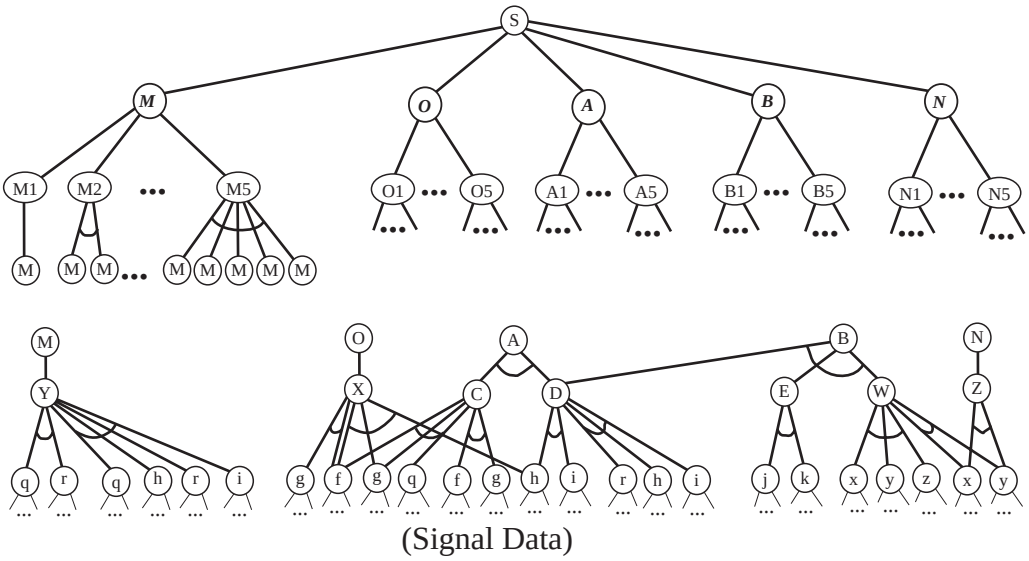


Figure 9.11. Interpretation Grammar $G2$

associated with the DVMT [Corkill, 1983] For each of the SNTs of G', M, O, A, B, N, we added nonterminals of the form A1, A2, A3, A4, A5 where the symbol could be any one of the SNTs and the number indicates the number of time-locations in a vehicle track. For example, one of the rules that was added is $A5 \rightarrow AAAAA$. We refer to this grammar as $G2$. Figure 9.11 shows the expanded grammar. The SNTs in this grammar are A, B, M, N, and O. As seen in the table, these additions to the grammar increased the expected cost of problem solving considerably. No bounding functions were used in this experiment.

9.4.3 Experiment 20

In this experiment, we added dynamic bounding functions with cost 0.01 throughout the grammar. The effects of this are quite dramatic. Specifically, the expected (and actual) cost of problem solving decreased by approximately 33% when compared with Experiment 19. It is also interesting to note that the percentage of correct answers found was 100.

9.5 Chapter Summary

This chapter introduces the concept of *potential* that is used to formally define the long-term benefits associated with a problem solving action. In many cases, the locally optimal control decision will not result in globally optimal problem solving. This occurs in situations where an operator on a search path does more than simply expand a state and extend a search path – where the operator actually *increases the information available to a problem solver regarding the interrelationships between partial solutions.* i.e., the operator increases the understanding of a partial solution's global significance. In these situations, the operator does not have to actually extend a search path in order to move the problem solver closer to termination, rather, the operator may alter the search space in some way that reduces the cost of problem solving (or increases the effectiveness of problem solving efforts) for some other set of operators. We will refer to this property of an operator as its *potential*.

The concept of potential allows the costs and benefits associated with meta-level operators to be directly compared with those associated with domain operators. Potential is calculated using the analytical tools enabled by the IDP and *UPC* formalisms. In general, the potential of an action is calculated by first examining what the long-term goals of the action are and then determining what the cost of problem solving would be if the long-term goals had already been achieved.

CHAPTER 10

REPRESENTING SOPHISTICATED CONTROL TECHNIQUES

In Chapter 4, several examples demonstrated how meta-level operators could be represented as production rules in IDP_I, the formal specification of a problems solver's available search operations. The examples, which included bounding functions and a form of goal processing intended to reduce overhead associated with redundant processing, were shown for simple grammars. In this section, two additional forms of meta-level operators, *preconditions* and *focus-of-control goal operators*, are presented using the vehicle tracking domain. These examples will be used in subsequent chapters to demonstrate some IDP/*UPC* analysis techniques that can be applied to sophisticated problem solvers.

The basic problem solving operators used in the vehicle tracking domain can be derived from the production rules of IDP_G. These operators are represented as production rules in IDP_I. For vehicle tracking, most of the production rules in IDP_I are be identical to their counterparts in IDP_G. The rules at the track level are slightly different from the corresponding production rules in IDP_G because they allow a track to be interpreted either forward or backward in time. This is in contrast to the generational rules which only generate data forward in time from a specified starting point. This is to allow more opportunistic search processes. Thus, given specific vehicle location data, the problem solver can attempt to form an interpretation by combining the data with other data that is either forward or backward in time. The problem solver is not restricted to going either one or the other direction.

The IDP_I production rules that define track processing operators are shown in Fig. 10.1. Note that IDP_I does not rely on the feature list convention but it does use a similar representation to indicate time dependencies. (The feature list convention is used in IDP_G to model the generation of data that exhibits relationships that span time and distance. It is not needed in IDP_I, which represents relationships between problem solving operators and intermediate and final results.) For example, rules 5 and 6 represent different operators that extend an existing track either forward or backward in time, respectively.

Rules 4, 9, 14, 19, 24, and 29 are used to construct a partial track from a vehicle location for different types of vehicles. Any vehicle location data can be used to start a track. There is no restriction that the initial partial track start at time 0 or on a region boundary. The problem solver can formulate a track starting anywhere along its length and then extending in either or both directions (i.e., forward and backward in time). Rules such as 5 and 6 are used to extend a partial track incrementally forward or backward in time. Partial tracks of arbitrary length can be combined using rules such as 7 and 8.

Note that the representation of operators in IDP_I is for analysis purposes only. When constructing an actual problem solver, it is not necessary to use this many distinct operators to represent different types of vehicle tracks. For practical implementations, it is reasonable to assume that there would be several "macro-operators" that would each correspond to multiple rules from IDP_I. For example, in an actual implementation, there might be an "extend forward"

1.	S	$\rightarrow$ Tracks
2.	Tracks	$\rightarrow$ Tracks Track
		$\rightarrow$ Track
3.	Track	$\rightarrow$ I-Track1
		$\rightarrow$ I-Track2
		$\rightarrow$ P-Track1
		$\rightarrow$ P-Track2
		$\rightarrow$ G-Track1
		$\rightarrow$ G-Track2
4.	I-Track1	$\rightarrow$ T1
5.	I-Track1	$\rightarrow$ I-Track1$_{[}t]$ T1$_{[}t-1]$
6.	I-Track1	$\rightarrow$ I-Track1$_{[}t]$ T1$_{[}t+1]$
7.	I-Track1	$\rightarrow$ I-Track1$_{[}t]$ I-Track1$_{[}t-1]$
8.	I-Track1	$\rightarrow$ I-Track1$_{[}t]$ I-Track1$_{[}t+1]$
9.	I-Track2	$\rightarrow$ T2
10.	I-Track2	$\rightarrow$ I-Track2$_{[}t]$ T2$_{[}t-1]$
11.	I-Track2	$\rightarrow$ I-Track2$_{[}t]$ T2$_{[}t+1]$
12.	I-Track2	$\rightarrow$ I-Track2$_{[}t]$ I-Track2$_{[}t-1]$
13.	I-Track2	$\rightarrow$ I-Track2$_{[}t]$ I-Track2$_{[}t+1]$
14.	P-Track1	$\rightarrow$ P-T1
15.	P-Track1	$\rightarrow$ P-Track1$_{[}t]$ P-T1$_{[}t-1]$
16.	P-Track1	$\rightarrow$ P-Track1$_{[}t]$ P-T1$_{[}t+1]$
17.	P-Track1	$\rightarrow$ P-Track1$_{[}t]$ P-Track1$_{[}t-1]$
18.	P-Track1	$\rightarrow$ P-Track1$_{[}t]$ P-Track1$_{[}t+1]$
19.	P-Track2	$\rightarrow$ P-T2
20.	P-Track2	$\rightarrow$ P-Track2$_{[}t]$ P-T2$_{[}t-1]$
21.	P-Track2	$\rightarrow$ P-Track2$_{[}t]$ P-T2$_{[}t+1]$
22.	P-Track2	$\rightarrow$ P-Track2$_{[}t]$ P-Track2$_{[}t-1]$
23.	P-Track2	$\rightarrow$ P-Track2$_{[}t]$ P-Track2$_{[}t+1]$
24.	G-Track1	$\rightarrow$ G-T1
25.	G-Track1	$\rightarrow$ G-Track1$_{[}t]$ G-T1$_{[}t-1]$
26.	G-Track1	$\rightarrow$ G-Track1$_{[}t]$ G-T1$_{[}t+1]$
27.	G-Track1	$\rightarrow$ G-Track1$_{[}t]$ G-Track1$_{[}t-1]$
28.	G-Track1	$\rightarrow$ G-Track1$_{[}t]$ G-Track1$_{[}t+1]$
29.	G-Track2	$\rightarrow$ G-T2
30.	G-Track2	$\rightarrow$ G-Track2$_{[}t]$ G-T2$_{[}t-1]$
31.	G-Track2	$\rightarrow$ G-Track2$_{[}t]$ G-T2$_{[}t+1]$
32.	G-Track2	$\rightarrow$ G-Track2$_{[}t]$ G-Track2$_{[}t-1]$
33.	G-Track2	$\rightarrow$ G-Track2$_{[}t]$ G-Track2$_{[}t+1]$

Figure 10.1. IDP$_I$ Production Rules for Interpreting Patterns and Tracks

macro-operator that would represent the problem solving actions corresponding to production rules 6, 11, 16, 21, 26, and 31 and that could be applied to any type of vehicle track.

The following sections present the definitions of preconditions and focus-of-control goal operators.

10.1 Representing Preconditions

Preconditions are meta-level operators that help focus search processes more effectively in sophisticated domains by performing some kind of preprocessing to more accurately judge the relative merits of executing an expensive problem solving action. For example, in a blackboard problem solver, a precondition might be executed before an operator is instantiated to determine if the data necessary to extend a partial result is present. This meta-level operation can be very effective in constructive search processes because each intermediate search state contains only local information about that particular state. Thus, if the rating of a potential search operation is heavily dependent on non-local information, the problem solver will not be able to accurately evaluate that action. The effectiveness of preconditions has been demonstrated in many interpretation systems including Hearsay II [Erman *et al.*, 1980] and the DVMT [Corkill, 1983].

Figure 10.2 illustrates the effect of using preconditions on the basic control architecture. The original control architecture is shown in Fig. 10.3 for comparison. During each control cycle, new states are created and the problem solver identifies which operators can be applied to the new states. Preconditions are applied to determine which operators are eligible for execution. The eligible operators are then rated and added to the queue of operators available for execution.

To clarify the use of preconditions, consider the production rule: $A \rightarrow B\,C\,D$. As suggested by Fig. 10.3, if the problem solver has generated a search state "B," the operator associated with the production rule would be eligible for execution whether or not the required states C and D had been generated. In situations where C and D had not been generated, the operator would fail to generate an A. In contrast, using preconditions, the operator would not be eligible for execution until all the required syntactic elements were present.

In more formal terms, as defined in Chapter 6, a problem solver is characterized by the four-tuple $< \mathcal{S}, \Omega, \omega, \Phi >$, where;

$\mathcal{S}$ = the *start state*; $\mathcal{S}$ is defined by the input data to the problem solver and the initial values of any relevant *Characteristic Variables* (CVs)[1].

Ω = the *base search space* with associated CVs and operators. Ω corresponds to the traditional notion of a search space. It is defined by CVs that specify the characteristics of individual states (including the *UPC* vectors), operators that map one state to another, and functions of CVs that define final states. In terms of an IDP, Ω is defined by the interpretation grammar.

ω = a set of *projections*, or *abstractions*, of the base search space, each with their associated CVs and operators. Final states are those from the base space that can be reached via mapping

[1] A state in the search space is defined by a set of CVs, where each CV_i is a characteristic variable with a corresponding attribute value.

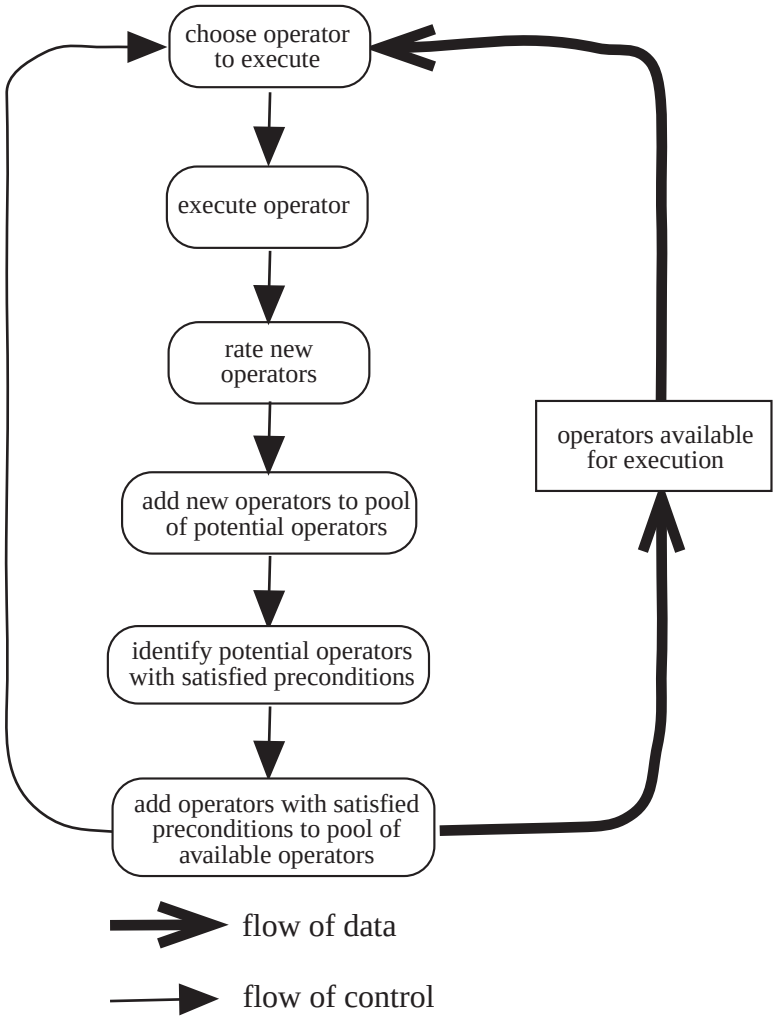


Figure 10.2. The Basic Control Cycle With Preconditions

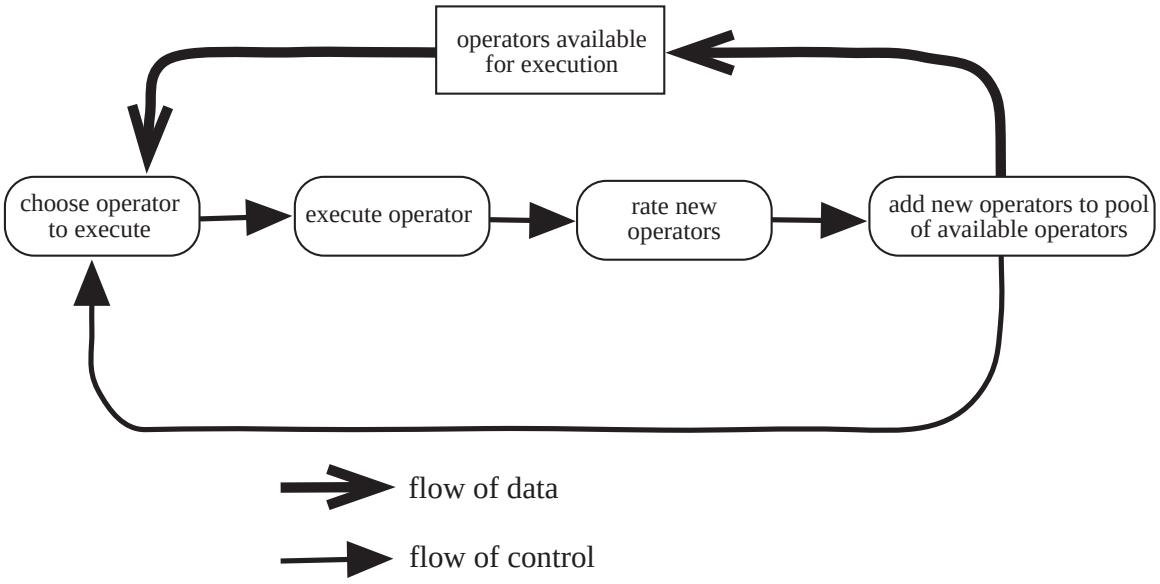


Figure 10.3. The Basic Control Cycle (Without Preconditions)

operators. A given search space projection ω_i is defined by two sets of operators, $OP_{(\Omega,\omega_i)}$ and $OP_{(\omega_i,\omega_i)}$. $OP_{(\Omega,\omega_i)}$ is the set of operators that project states from Ω to states in ω_i. $OP_{(\omega_i,\omega_i)}$ is the set of operators that map states in ω_i to other states in ω_i. As with Ω, each state in a projected search space is characterized by a set of CVs and a set of *UPC* vectors.

Φ = a set of ***mapping functions*** from projection spaces back to the base search space. Functions in Φ can be thought of as the mechanisms that map constraints from an abstract space, ω_i, back to the base search space, Ω. This can be done by creating new states in Ω, or by modifying existing states.

To represent preconditions using the IDP formalism, we will define a projection space of satisfied precondition states. This space is created by operators of the form: $A^{op}_{[B,C,D]} \rightarrow$ B C D. Thus, a symbol with a superscript *op* will represent a state created by a successful precondition operator. *op* will correspond to the operator that will map the abstract state back to the base space. For clarity, the subscripted brackets will include the required syntactic elements used to create the precondition state. A production rule of this form will be created for each precondition operator. More formally, we say that the set of projection operators from the base space to the precondition abstraction space, $OP_{(\Omega,\omega_{precondition})}$, is made up of precondition operators.

In addition, a mapping operator from the precondition projection space back to the base space will have the form: A $\rightarrow A^{op}_{[B,C,D]}$. This operator will correspond to the original operator except that it can only be applied to the states in the precondition space. The original operator will be replaced by these two rules. Formally, the set of mapping operators from the precondition abstraction space back to the base space is represented as $OP_{(\omega_{precondition},\Omega)} \in \Phi$.

For the moment, we will not define any operators that create new states in the precondition space from states already in the space. Formally, this means that the set $OP_{(\omega_{precondition},\omega_{precondition})}$ is empty.

50. P-T1 → T1 T2
51. P-T2 → T2 T2
52. G-T1 → GT1 T1
53. G-T2 → GT2 T2
54. T1 → V1 N
55. T2 → V2 N

Figure 10.4. IDP$_I$ Production Rules for Interpreting Vehicle and Track Locations

P.50. P-T1$^{op_{50}}_{T1,T2}$ → T1 T2
P.51. P-T2$^{op_{51}}_{T2,T2}$ → T2 T2
P.52. G-T1$^{op_{52}}_{GT1,T1}$ → GT1 T1
P.53. G-T2$^{op_{53}}_{GT2,T2}$ → GT2 T2
P.54. T1$^{op_{54}}_{V1,N}$ → V1 N
P.55. T2$^{op_{55}}_{V2,N}$ → V2 N

Figure 10.5. Precondition Operators for Vehicle and Track Locations

Figures 10.4, 10.5, and 10.6 show examples of preconditions, mapping functions, and the original production rules for the vehicle tracking domain. Figure 10.4 shows the base space rules for interpreting vehicle, pattern, and ghost track locations. The production rules in Fig. 10.5 are the projection operators corresponding to the base space rules. The operators that map states from the precondition abstraction space back to the base space are shown in Fig. 10.6.

10.2 Representing Goal Processing

A variety of goal processing systems have been developed to focus problem solving activities more efficiently and to reduce redundant processing [Erman *et al.*, 1980, Lesser *et al.*, 1989a,

M.P.50. P-T1 → P-T1$^{op_{50}}_{T1,T2}$
M.P.51. P-T2 → P-T2$^{op_{51}}_{T2,T2}$
M.P.52. G-T1 → G-T1$^{op_{52}}_{GT1,T1}$
M.P.53. G-T2 → G-T2$^{op_{53}}_{GT2,T2}$
M.P.54. T1 → T1$^{op_{54}}_{V1,N}$
M.P.55. T2 → T2$^{op_{55}}_{V2,N}$

Figure 10.6. Mapping Operators for Vehicle and Track Locations – From Precondition Space to the Base Space

Corkill and Lesser, 1981]. In Chapter 4, we showed how a specific form of goal processing used in the DVMT to eliminate redundant local activity [Corkill, 1983] could be represented in IDP_I grammars.

Another form of goal processing was developed and applied in the DVMT to focus problem solving activity more effectively. Focus of control goal processing was used to increase the rating of problem solving activities that were related to extending highly-rated partial results. For example, given a partial track with a high-rating, the focus of control goal processing would increase the ratings of operators that would generate the low-level data needed to extend the track. This would cause the low-level data to be generated sooner than it otherwise would have been, allowing the problem solver to extend the highly-rated partial track hypothesis. Figure 10.7 shows the architecture of a problem solver that uses focus-of-control goal processing and preconditions. Goal processing is represented as the subprocess labeled, "rerate operators based on goals." As shown, this process takes information from the pool of available operators, modifies it, and updates the appropriate elements of the pool.

To understand the benefits associated with this form of goal processing, consider a problem solver that combines it with dynamic pruning operators. Assuming that the pruning operators eliminate paths from consideration by comparing them with existing full interpretations, the more quickly the problem solver can construct a highly-rated full interpretation, the less total work is likely to be required. Given a partial track with a high-rating, it may be desirable to extend the track as quickly as possible to a full solution to maximize its pruning potential. However, if the data needed to extend the partial track is not available, the partial track cannot be extended. Focus-of-control goal processing is used to increase the rating of intermediate operators that will generate data needed to extend the partial track.

From a more general perspective, focus-of-control goal processing is a mechanism that allows a problem solver to selectively rerate problem solving operators based on their relationships with other, dynamically changing operators. As discussed in Chapter 2, one of the characteristics of sophisticated domains is that partial search paths are often related and the structure of these relationships can sometimes be exploited to increase the efficiency of problem solving. Exhaustive processing of these relationships during each problem solving cycle is infeasible because the number of relationships is combinatoric in the number of partial search paths and each relationship could involve arbitrarily complex processing. Focus-of-control goal processing restricts the search for and processing of relationships to cover only those relationships for which it is cost effective. This is a very general technique that is likely to have applicability in a wide range of domains.

Example goal processing operators are shown in Fig. 10.8. Operator G.4.1 is used to raise the ratings of operators that will generate data needed to extend a partial I-Track1 forward in time. Similarly, G.4.1 is used to extend a partial I-Track1 backwards in time. Rules G.5.1 and G.5.2 function similarly for I-Track2 partial results.

Like the precondition operators, the goal processors define a projection space. Example mapping operators for this space are shown in Fig. 10.9. The mapping operators are applied to the states of the goal projection space. The $\emptyset$ notation on the LHS of the production rules indicates that no new states are generated in the base space. Instead, the ratings for instantiated operators is modified to reflect the information contained in the goal state.

For example, in a vehicle monitoring domain, the result of approximate processing could be the generation of a goal to extend a partial track into an area where there is noise. This is illustrated in Fig. 10.10. (Figure 10.10 and the following discussion is based on the grammar introduced in Figures 4.25, 4.27, 4.29, and 4.30.) In order to extend the partial track

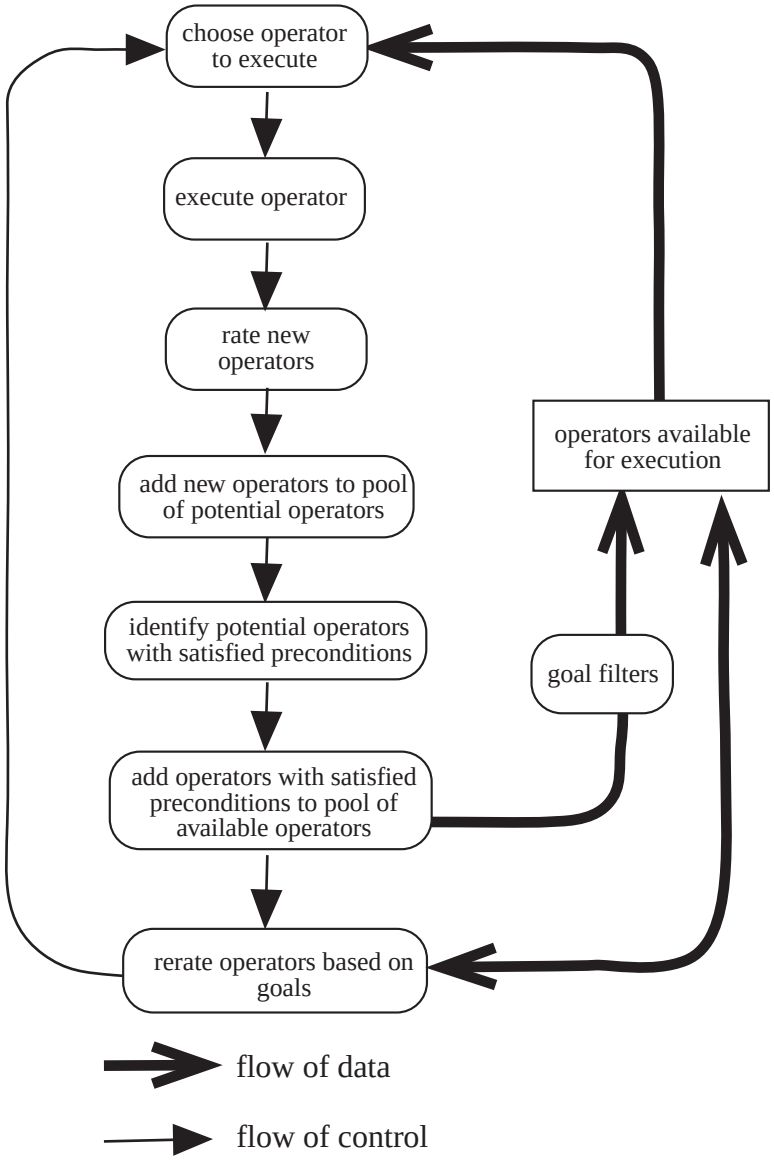


Figure 10.7. The Basic Control Cycle With Preconditions and Goal Processing

G.4.1. Goal-TL1$_{[t+1]}$ $\rightarrow$ I-Track1$_{[t]}$
G.4.2. Goal-TL1$_{[t-1]}$ $\rightarrow$ I-Track1$_{[t]}$
G.5.1. Goal-TL2$_{[t+1]}$ $\rightarrow$ I-Track2$_{[t]}$
G.5.2. Goal-TL2$_{[t-1]}$ $\rightarrow$ I-Track2$_{[t]}$

Figure 10.8. Meta-Level Operators for Focus-of-Control Goal Processing

M.G.5. $\emptyset \rightarrow$ Goal-TL1$_{[t]}$
M.G.6. $\emptyset \rightarrow$ Goal-TL2$_{[t]}$

Figure 10.9. Mapping Operators for the Goal Projection Space

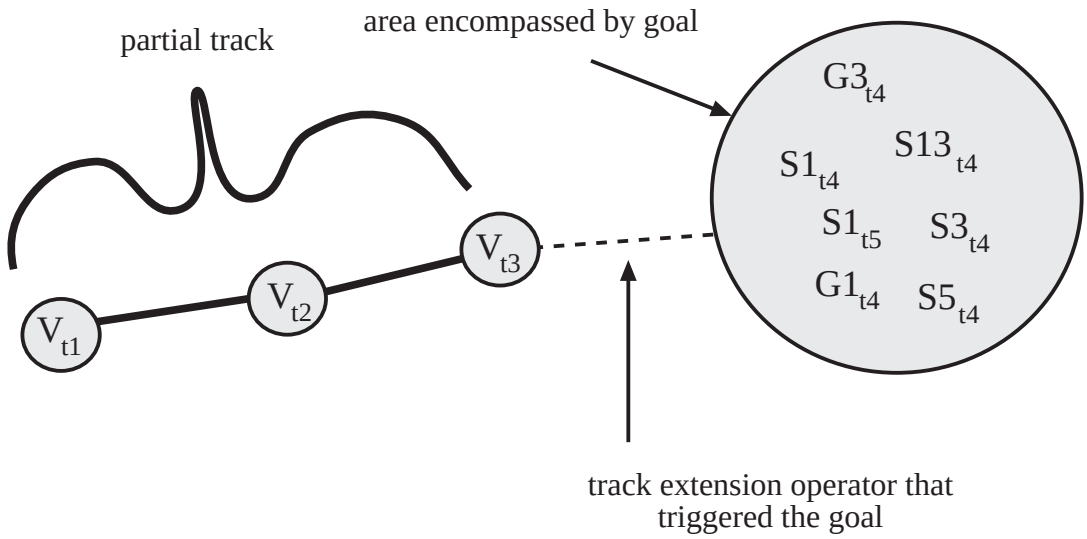


Figure 10.10. Example of the Use of Goal Processing in Vehicle Tracking

I-Track1$_{1-3}$, the problem solver needs to first generate one or more track location level results, T1$_4$, in the area indicated by the noise. However, because of the noise, the low-level operators that are needed to generate the track location level result have low ratings compared with other operators and, consequently, the required partial results are not available.

To generate the required track location result, the problem solver creates a goal, Goal-TL1$_{4,a}$, where the subscript indicates the characteristics of the partial results to which the goal is applicable. In this example, the subscript 4 corresponds to the time characteristics and the subscript "a" corresponds to a representation of the area encompassed by the goal. In the experimental problem solver, the area is specified by a center point and a radius. For more details, see Appendix C.

The goal is mapped to the base space as described above. In the experimental problem solver, the mapping mechanism is implemented as two processes. In the first process, which is represented by the subprocess labeled "rerate operators based on goals" in Fig. 10.7, the rating of all the operators encompassed by the goal are examined and are raised to the *goal rating*. (If an operator's current rating exceeds the goal rating, it is not modified.) In the situation shown in Fig. 10.10, the *goal rating* would be equivalent to the rating of the operator attempting to extend the partial track, op$_4$. Figure 10.11 shows the effects of mapping the goal back to the base space. The figure contains three sections, a goal description, representations of a set of states and operators before goal processing, and a set of states and operators after goal processing.

The goal description includes the goal's name, its time and location attributes, the potential solutions that can be reached on paths from goal, and the goal's rating. The state/operator

Goal Description

Goal	Time/Location	Potential Solutions	Rating
Goal-TL1	4/a	I-Track1	0.8

States/Operators Before Goal Processing

State	Time/Location	Operator	Potential Solutions	Rating
S1	4/a	op_{23}	I-Track1, P-Track1, G-Track1	0.5
S1	5/a	op_{23}	I-Track1, P-Track1, G-Track1	0.5
S3	4/a	op_{23}	I-Track1, P-Track1, G-Track1	0.6
S5	4/a	op_{24}	I-Track1/2, P-Track1/2, G-Track1/2	0.9
S13	4/a	op_{26}	I-Track2, P-Track2, G-Track2	0.4
G1	4/a	op_{19}	I-Track1, P-Track1, G-Track1	0.6
G3	4/a	op_{19}	I-Track1, P-Track1, G-Track1	0.7
G3	4/a	op_{20}	I-Track2, P-Track2, G-Track2	0.7

States/Operators After Goal Processing

State	Time/Location	Operator	Potential Solutions	Rating
S1	4/a	op_{23}	I-Track1, P-Track1, G-Track1	0.8
S1	5/a	op_{23}	I-Track1, P-Track1, G-Track1	0.5
S3	4/a	op_{23}	I-Track1, P-Track1, G-Track1	0.8
S5	4/a	op_{24}	I-Track1/2, P-Track1/2, G-Track1/2	0.9
S13	4/a	op_{26}	I-Track2, P-Track2, G-Track2	0.4
G1	4/a	op_{19}	I-Track1, P-Track1, G-Track1	0.8
G3	4/a	op_{19}	I-Track1, P-Track1, G-Track1	0.8
G3	4/a	op_{20}	I-Track2, P-Track2, G-Track2	0.7

Figure 10.11. Results of Mapping a Goal Back to the Base Space

representations shows a set of states, the time/location characteristics of the states, the operators instantiated to process the states, the operator ratings, and the potential solutions[2] that can be reached on paths starting from the state and including the operator. The state subscripts indicate the time and location characteristics of the state.

As shown in the figure, op_{23} applied to states S1 and S3 with Time/Locations that are encompassed in the goal's Time/Location have their ratings increased to be the same as the goal rating. op_{23} applied to state S1 with Time characteristic 5 does not have its rating increased because the state is not encompassed by the goal. op_{24} applied to state S5 does not have its rating changed because, even though it is encompassed by the goal, its rating is already higher than the goal rating. op_{26} applied to state S13 does not have its rating increased because the state is not a member of the goal's component set. The goal's component set only includes

[2]The potential solutions, which we refer to as Solution nonterminals (SNTs) correspond to corresponding to the entries in each operator's *UPC* utility vector. As described in Chapter 6, each operator has an associated utility vector that lists the expected utility of each of the SNTs, or potential solutions, that can be reached on paths starting with the path segment(s) that are to be created by the operator.

states that are included in paths to the SNTs I-Track1, P-Track1, and G-Track1. op_{19} applied to state G3 has its rating increased, but op_{20}, applied to the same state, does not. This is because op_{20} is not on a path to one of the SNTs in the goal's Potential Solutions set.

The second form of processing used to implement goal mapping is very similar to the first form, but it focuses on operators that are not yet created when the original goal is created and, consequently, do not have their ratings increased by the first process. The second process uses filters to raise the ratings of new operators as they are generated and added to the queue. This is illustrated in Fig. 10.7 by the processes labeled "Process Goal Filters." As implied by the figure, each new operator that is created passes through each goal filter. The rerating mechanism used by a goal filter is identical to that described above only it is applied to operators as they are created.

As with the precondition abstraction space, there are no operators in the set $OP_{(\omega_{goal},\omega_{goal})}$. Thus, in this example, there are no operators that create new goal states from existing goal states. However, it is easy to imagine such operators and to see how they have been incorporated into sophisticated problem solving systems. These operators are described in the next section.

10.3 Representing Clustering and Abstract Level Processing

Many research projects have demonstrated the benefits of aggregating data for subsequent approximate processing or for generating approximate solutions based on preliminary problem solving and then using these results to increase the efficiency of subsequent processing. Examples of these in interpretation domains include the *clustering mechanisms* and associated abstractions used by Durfee and Lesser in [Durfee and Lesser, 1986] and similar mechanisms developed in Hearsay II [Erman *et al.*, 1980]. In addition, these strategies are commonly applied to other domains with similar results, such as those demonstrated by Knoblock in planning tasks [Knoblock, 1991b].

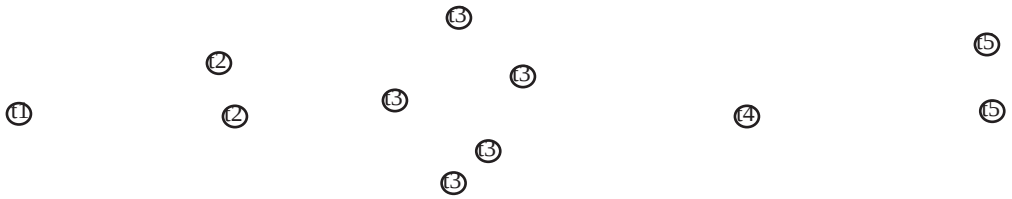
The following sections demonstrate how the IDP formalism can be used to represent abstract states formed by approximations or aggregations and the mechanisms that are used to create, process, and refine abstract states.

10.3.1 Generating Abstract Clusters Through Aggregation

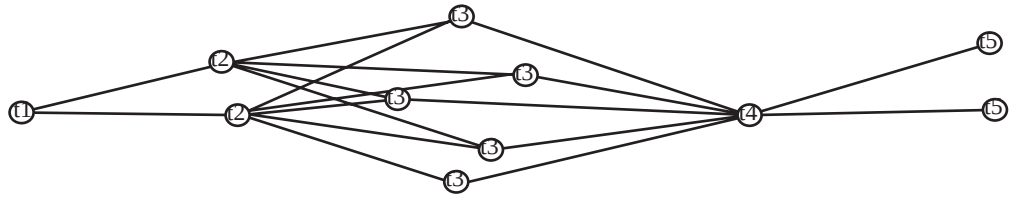
Approximating data into abstract clusters is a strategy that attempts to efficiently apply constraints to aggregations of data and it is particularly useful in domains with large quantities of noise. Instead of processing each individual piece of data, a problem solver can aggregate data into an abstract unit and then process the unit as a whole. For example, if the initial input data is very noisy, and if the noise and the correct data have very similar characteristics, they can be clustered into a single datum with characteristics encompassing both [Decker *et al.*, 1990].

Figure 10.12 illustrates a situation with a number of partial results where each partial result has an attribute for time and location. Shown are the clusters formed by ignoring one of the attributes - vehicle location. This form of approximation can also be thought of as replacing a single value with a range of values. In this case, specific x and y locations are replaced by a center point and radius (which will subsequently be referred to as "R") that encompasses all the aggregated data.

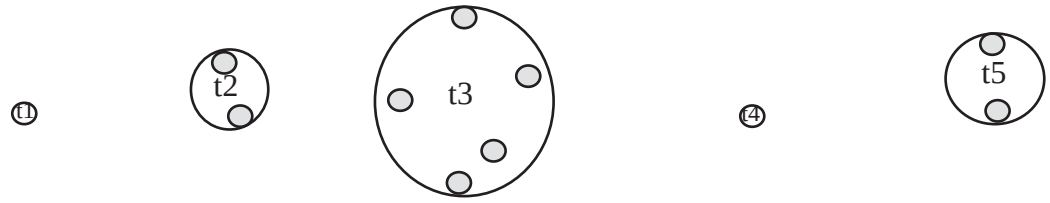
These approximations result in the definition of a more abstract state-space. The problem solver can then continue the interpretation process in this projection space to create complete interpretations that are abstractions of potential complete interpretations in the base space.



a. original data

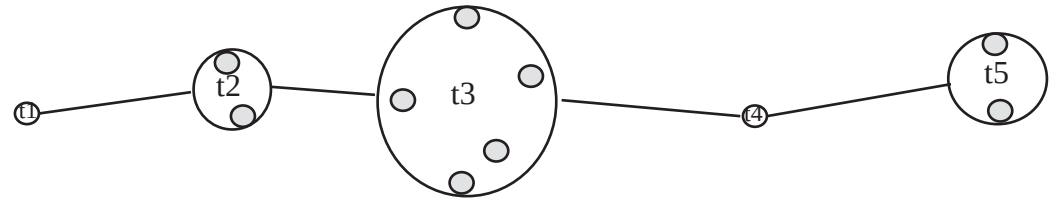


b. solutions constructed with original data



c. result of clustering original data

.



d. solutions constructed with clustered data

Figure 10.12. Abstract States and Projection Space Solutions Constructed from Approximate Data

This is shown in Figures 10.12.c and 10.12.d. It must be stressed that these projection space solutions are not guaranteed to contain base space solutions. Representing the processes used to create projection space solutions using cluster abstractions and mapping them back to the base space is discussed in Chapter 10.3.3.

C.1. $S1^p \rightarrow S1_{[f_1,t]} \ldots S1_{[f_n,t]}$
C.2. $S2^p \rightarrow S2_{[f_1,t]} \ldots S2_{[f_m,t]}$
...
C.x $Sx^p \rightarrow Sx_{[f_1,t]} \ldots Sx_{[f_m,t]}$

Figure 10.13. Clustering Operators for Signal Data

The clustering operators available to a problem solver define a projection space and are represented in the IDP formalism with the form shown in Fig. 10.13. As shown in the figure, the abstract clustered data is represented with a superscript, p, that indicates a precision level, which is defined below. Each cluster comprises an arbitrary number of partial results of the same type. Thus, as shown in rule C.1, the abstract state $S1^p$ is formed from some number of S1 partial results from the base space and exists in the projection space defined by the precision metric p. In the vehicle tracking domain, the partial results that are aggregated into a cluster can have different feature list characteristics that define "event class," but they must share the same time characteristic. In other words, a signal cluster can comprise data from different signal classes.

As suggested by Fig. 10.13, the clustering operators specify a projection space and a set of operators $OP_{(\Omega,\omega_p)}$, where p, a partial result's *precision*, is a specification of the level of abstraction of a given projection space. As shown in Fig. 10.14, this definition effectively expands the dimensionality of the blackboard along an axis corresponding to precision. In the vehicle tracking domain, a partial result's precision is a function of its size, R (which indicates the radius of the area that encompasses all the partial results included in a cluster), the number of different types of partial results, E (which indicates the number of different elements in the cluster), and the variance of the partial result's likely location within R.

In the IDP/*UPC* formalism, the semantics associated with approximate data must be consistent with those associated with precise data so that meaningful combinations of the two can be constructed through projection and mapping functions. If approximate data has a significantly different semantic interpretation than precise data, it might not be possible to incorporate both into a single perspective of problem solving.

For example, several acoustic signals might be clustered into a single partial result that encompasses not only the area of the sensed data, but areas where no data was sensed as well. The new data cluster has several interpretations. It can take on an *existential* interpretation, such as "there is some support for a signal source *somewhere* in this area", or it can take on a *universal* interpretation, such as "there is some support for a signal source at *every* point in this area." If some problem solving activity involves checking for physical overlap among data, then the existential and universal interpretations will produce distinct results. In particular, determining

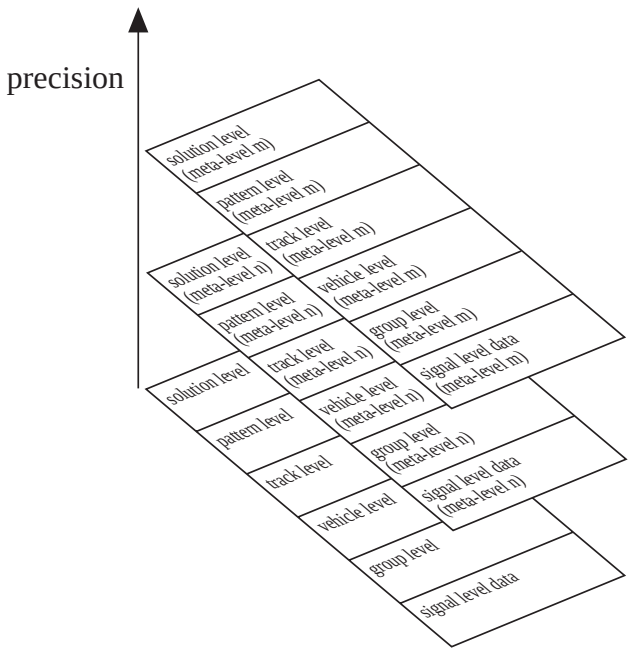


Figure 10.14. Blackboard Meta-Levels Defined by Precision Metric

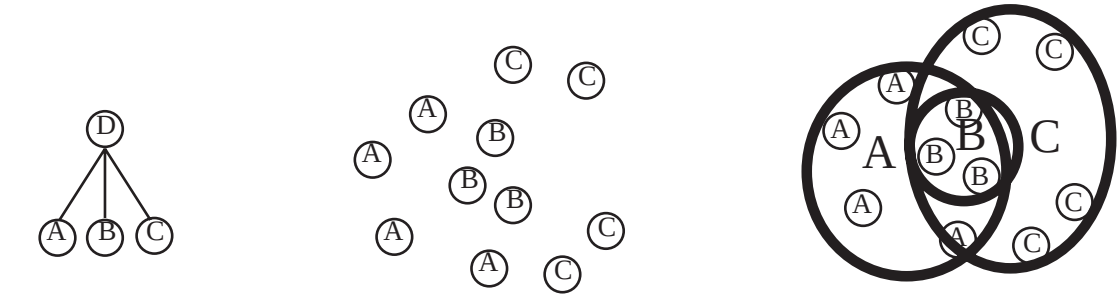


Figure 10.15. Data Approximation and Loss of Certainty

whether or not an existentially interpreted datum and a universally interpreted datum overlap might be semantically inconsistent. On the other hand, if precise and approximate data are readily interchangeable, then the universal problem solving perspective of the IDP/*UPC* formalism is preserved.

These issues are illustrated in Fig. 10.15. In the figure, an interpretation of a D requires overlapping A, B, and C. However, when the data is clustered, the clusters overlap, indicating that it is possible to generate an interpretation of a D. With a universal interpretation of the data clusters, it might be reasonable to consider the D a correct interpretation. However, with an existential interpretation of the data clusters, the problem solver will recognize that there is a possibility that, even though the clusters overlap, the precise data might not. Thus, additional refinement will be required to map the results back to the base space.

An existential interpretation of data has been adopted for the IDP/*UPC* framework. An *exact*, or precise, hypothesis is represented with a range attribute specified by a single point, an event class set with cardinality one, and a precision of zero[3]. A *cluster* hypothesis has a range, R, defined as a convex region encompassing all the component locations, an event class $E = \cup$(component event classes), and a precision $= f_p$(size of R, $\mid E \mid$, f_v), where f_v is the variance of the clustered hypotheses' locations within R.

Figure 10.16 shows several examples of approximated data. In this example, f_p = ec * size * s(x) * s(y), where ec is the cardinality of E, size is the size of the cluster, and s(x) and s(y) are the standard deviations of the x and y location coordinates. Part (a) is an exact location hypothesis with precision zero; the size of the range spanned is zero and the vehicle's variance within that range is also zero. The location cluster shown in (b) is less precise than the exact location hypothesis because nine distinct vehicle hypotheses have been aggregated into one. The result is a cluster spanning a larger range and having a non-zero variance within that range. Even though they have similar sized ranges, (b) is more precise than the location cluster shown in (c) because the variance of vehicle locations within (c) is much greater than in (b). (d) highlights the adverse impact wide distributions of data can have on a cluster's precision. Despite having a slightly smaller range,(d) is less precise than either (b) or (c) because of the large variance of vehicle locations it encompasses. (e) shows the adverse effects of clustering multiple event classes. (e) is similar in size and variance to (c) but it is much less precise because it encompasses multiple event classes. (f) and (g) show how quickly precision is reduced when widely distributed data with multiple event classes is clustered.

When a location clustering operator, such as those defined in Fig. 10.13, is invoked, it generates clusters using an *exclusive, intrinsic, hierarchical* method as shown in Fig. 10.17. This algorithm is incorporated into the experimental problem solver described in Chapter 12 and the general approximate processing architecture in Appendix C.

10.3.2 Generating Abstract States Through Search and Knowledge Approximation

Two general approximation strategies can be used to generate abstract states; search approximation and knowledge approximation. These strategies are defined and discussed in [Decker *et al.*, 1990, Lesser *et al.*, 1988a]. The specific techniques described here are *eliminating corroborating support* and *level hopping*. These techniques are also defined and discussed in [Decker *et al.*, 1990, Lesser *et al.*, 1988a]. Both general strategies are based on reducing the amount of the search space that is explored in generating an interpretation. As a consequence, the resulting interpretation is considered an abstract state in a projection space.

Given the IDP/*UPC* framework, it is possible to formalize the representation of each of these techniques. This formalization can then be used for design and analysis. This capability is based on the extent to which a domain and the associated problem solver can be represented with the IDP/*UPC* framework. In general, the production rules of IDP$_g$ explicitly represent all the natural constraints of a problem domain. Specifically, as shown in Fig. 10.18, constraints exist among the elements of a RHS of a grammar rule and between the LHS of a grammar rule and all the elements of a RHS. Furthermore, constraint relationships are transitive. Thus, constraints are propagated from an LHS element to all its descendants. Thus, in the IDP formalism, approximate processing can be viewed as a form of constraint relaxation. The

[3] Because the precision measure is unbound, the problem solver uses an inverted precision scale. Thus, exact data has precision zero, and greater values of precision indicate less precise data.

Precision (ec)(size) + s + s

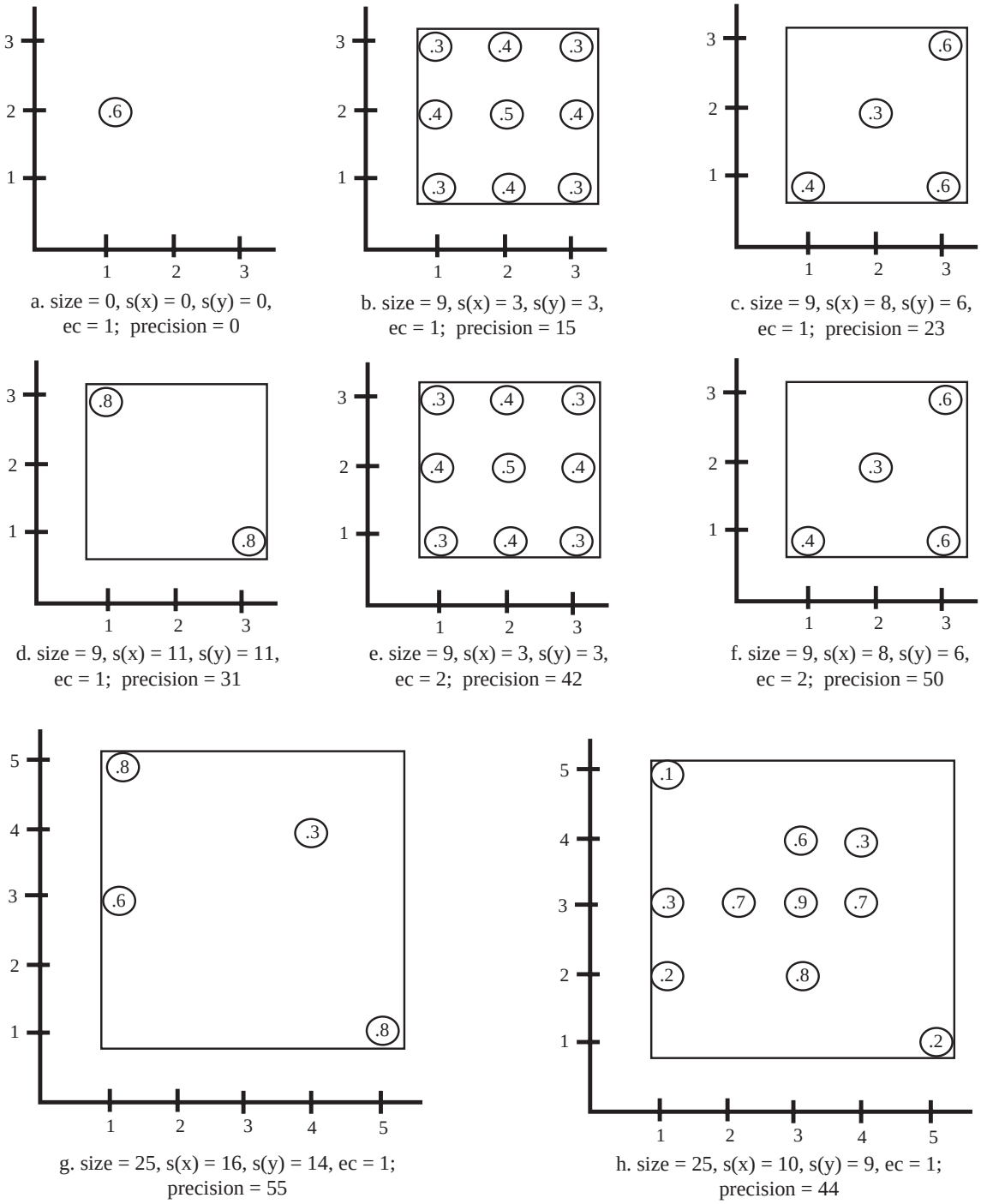


Figure 10.16. Examples of Precision Metric

1) GIVEN region R, cluster size limits δ and desired cluster precision τ

Where τ defines the maximum number of clusters to be returned.

2) FIND all hyps overlapping R and put them in *hyplist*

SORT *hyplist* by belief

SET j=1

FOR each hyp, h_i, in *hyplist* DO

SET $CS_j = (h_i)$

FOR every hyp, h_k, $k \neq i$, in *hyplist* DO

IF h_k is within δ of h_i

THEN ADD h_k to CS_j

ENDIF

INCREMENT j

(Note:step 2 forms "preclusters". This step isn't necessary, but it reflects the algorithm used in the example.)

3) REPEAT (UNTIL number of clusters = τ)

FOR all clusters, CS_i, DO

DETERMINE μ_i and σ_i

FOR all clusters, CS_j, DO

FOR all clusters, CS_k, DO

$tolerance_{j,k}$ = DISTANCE(σ_j, σ_k) / σ_j +

DISTANCE($\sigma_j, \sigma k$) / σ_k

COMBINE CS_j and CS_k such that $tolerance_{i,j}$ is minimized

ENDREPEAT

Figure 10.17. Cluster Generation Algorithm

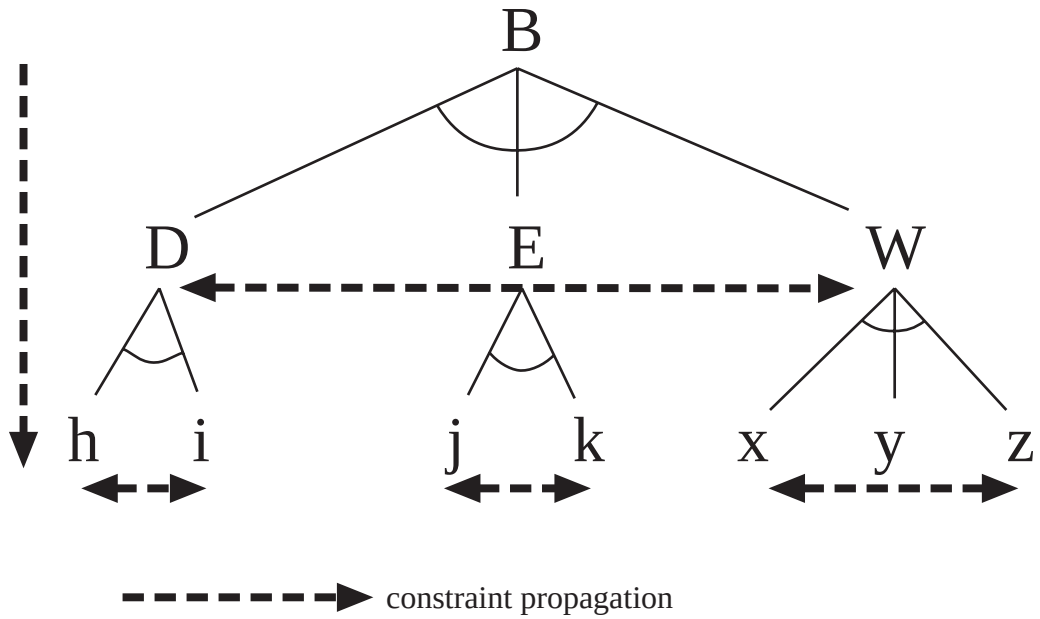


Figure 10.18. Domain Constraint Propagation

formal representation of approximation techniques that is described in the following sections is therefore similar to Knoblock's approach in that it is based on dropping or simplifying constraints [Knoblock, 1994].

More specifically, eliminating corroborating support is an approximation strategy we have defined for interpretation domains that avoids some of the processing that would otherwise be used to generate a precise interpretation. If a problem solver can identify processing that is based on highly probable events or indicators, it might be able to eliminate alternative work with predictable effects. For example, in Fig. 10.19, an approximation of H is defined by modifying the grammar rules to eliminate processing of A, B, C, and F. This can be a very effective approximation technique if there is a strong causal relationship between the remaining components, D and E, and H. i.e., if H is strongly implied by D and E. Chapter 12 discusses techniques for designing effective approximation operators that eliminate corroborating support.

The IDP definition of abstract operators that eliminate corroborating support have the form shown in Fig. 10.20. These examples are of vehicle location level production rules that are approximations of rules 19 and 20 from the vehicle interpretation grammar. The superscript indicates that they define a projection space *ecs*. To be precise, these operators define the set $OP_{(\Omega,\omega_{ecs})}$. In ECS.19, the corroborating support from G3 and G7 is eliminated. In ECS.20, the corroborating support from G3 and G1 is eliminated.

In contrast to search approximations, knowledge approximations simplify or eliminate the constraints used by a problem solver. For example, a problem solver may have a very accurate algorithm for determining some solution attribute. However, this algorithm may be computationally expensive. If the problem solver can predict the effects of using a less costly, under constrained algorithm, it is possible to define a projection space based on this algorithm. In general, if the partial results generated by a knowledge approximation are a superset of the partial results that would be generated with the corresponding exact knowledge, *then the*

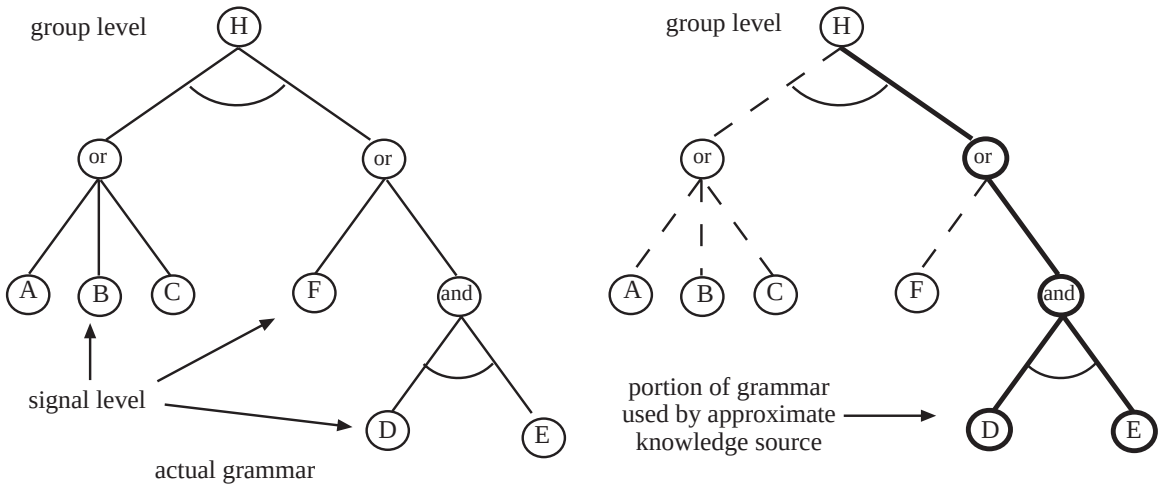


Figure 10.19. IDP Representation of Approximating Search - Eliminating Corroborating Support

$$\text{ECS.19} \quad V1^{ecs}_{[f]} \rightarrow G1_{[f]}$$
$$\text{ECS.20} \quad V2^{ecs}_{[f]} \rightarrow G8_{[f]}$$

Figure 10.20. Abstract Operators Based on Eliminating Corroborating Support

knowledge approximation is a candidate for use as a control mechanism. This is also the case if it is true "most" of the time.

Figure 10.21 illustrates *level-hopping*, an other form of approximate processing we have defined for interpretation domains. Level-hopping takes constraints that would normally span multiple levels of the blackboard in the base space and simplifies and compresses them into a single operator. In Fig. 10.21, the intermediate level of processing has been eliminated and the multiple levels are compressed into a single-step operation.

The general effect of level-hopping is shown in Fig. 10.22. As shown, level-hopping does not eliminate from consideration any supporting data. Instead, it eliminates the constraint processing that would have generated the intermediate results W and Y at the group level. As a consequence, the abstract Z that is created is underconstrained and, as such, it is a superset of the vehicle level interpretations that would have been generated in the base space.

In the IDP formalism, level-hopping operators have the form shown in Fig. 10.23. These examples are level-hopping operators corresponding to rules 19 and 20 from the vehicle interpretation grammar. The superscript indicates that they define a projection space *lh*. To be precise, these operators define the set $OP_{(\Omega,\omega_{elh})}$. These operators are constructed by fully expanding the group level results that would be used to generate V1 and V2 results in the base space.

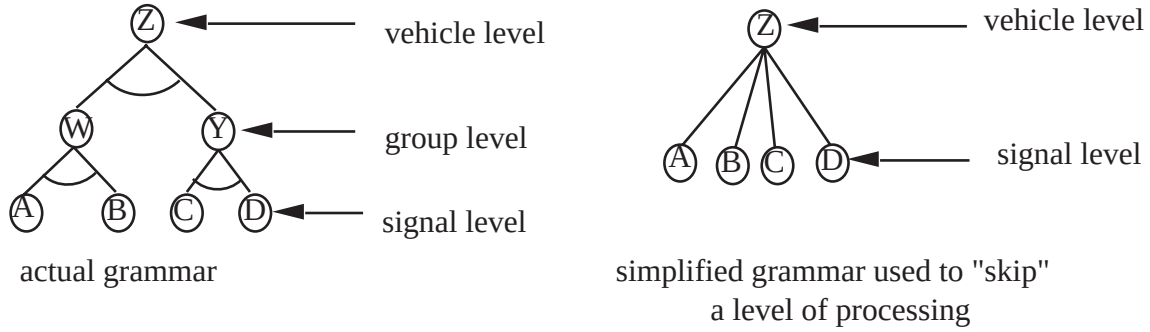


Figure 10.21. IDP Representation of Level Hopping in the Vehicle Tracking Domain

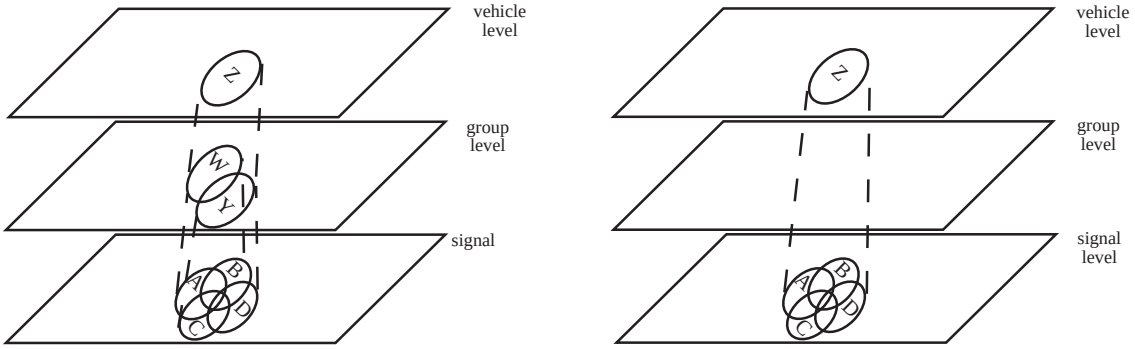


Figure 10.22. Illustration of Level Hopping

LH.19 $V1^{lh}_{[f]} \rightarrow S1_{[f]}\ S2_{[f]}\ S5_{[f]}\ S7_{[f]}\ S11_{[f]}\ S15_{[f]}$

LH.20 $V2^{lh}_{[f]} \rightarrow S5_{[f]}\ S6_{[f]}\ S7_{[f]}\ S13_{[f]}\ S14_{[f]}\ S17_{[f]}\ S18_{[f]}$

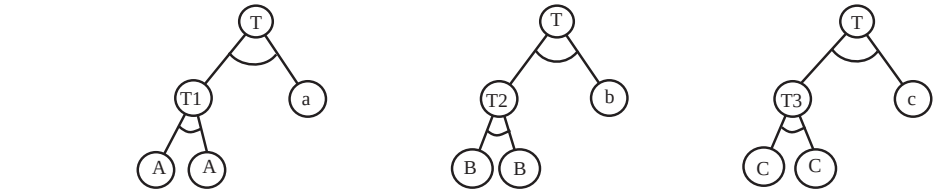
Figure 10.23. Abstract Operators Based on Level Hopping

10.3.3 Processing Abstract Clusters

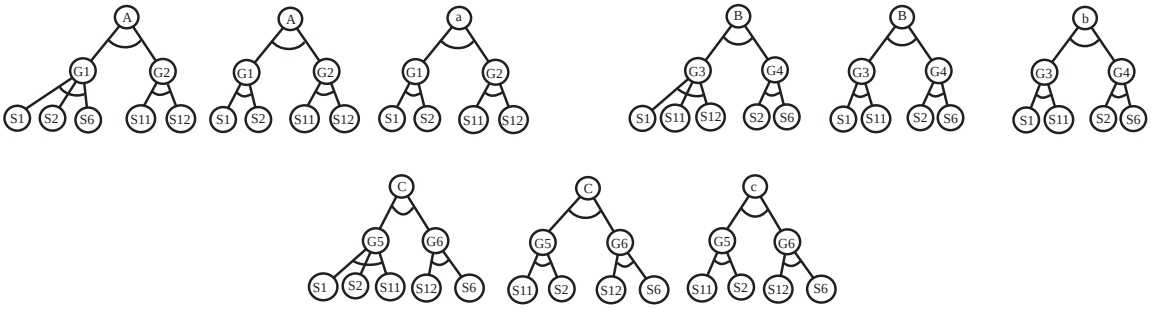
The important consideration relative to the preceding sections is whether or not the abstractions generated by aggregating data, approximating search, and approximating knowledge can somehow be used to reduce the cost of problem solving. As mentioned previously, it has been demonstrated that using approximations to developing a general understanding of a particular problem instance and then using that understanding to more effectively control problem solving can be an effective problem solving technique. Approximations can also be used for real-time processing [Decker *et al.*, 1990]. Such use would complicate IDP representations by setting a fixed time constraint on an interpretation problem which would affect the precision and credibility of possible results. This would change the optimality criterion and the associated decision strategy.

AP.14	$T1^{p}$	$\rightarrow V1^{p}$
AP.4	I-Track^{p}1	$\rightarrow$ I-Track1$^{p}_{[}t]$ T1$^{p}_{[}t-1]$

Figure 10.24. Abstract Operators for Processing Approximations



a. grammar rules for forming tracks



b. grammar rules for vehicle, group, and signal locations

Figure 10.25. Approximate Processing IDP/*UPC* Example

With respect to interpretation domains, Chapter 12 discusses IDP/*UPC*-based techniques for designing problem solving operators and control strategies that use these approximations. This section defines how the approximate processing techniques are represented.

As discussed in more detail in Appendix C, the IDP/*UPC* framework relies on an existential perspective of data. This allows the framework to extend the unified representation of base space processing and control to encompass the approximations defined in the preceding sections. Specifically, in the IDP/*UPC* framework, approximate processing operators have the form shown in Fig. 10.24. In the example form, the precision metric p is used to specify *the abstraction level at which the operator is applied*. Thus, the abstract operators are merely extensions of the base space operators.

Intuitively, this representation works because the constraints that are applied in the abstract space are identical to those that are applied in the base space. The only difference is that in the abstract space, the constraints are applied to CVs that are specified as ranges rather than precise values. In fact, as will be discussed in Appendix C, the experimental testbed was modified to process range data rather than precise data.

10.3.4 Mapping Abstract Results to the Base Space

There are a variety of techniques for mapping the results of abstraction level processing back to the base space. In general, all these techniques involve manipulating *UPC* values of search paths. In this section, we describe several of the techniques we have implemented and discuss their relation to existing problem solving systems.

The simplest form of mapping, *rating modification mapping*, is to use the results of approximate processing to modify the ratings of operators in order to bias, or focus, the

problem solver's actions toward a particular set of search paths. For example, as discussed in Chapter 10.2, goal processing can be used to increase the ratings of lower-level actions so that intermediate results required by a high-level action are generated more quickly. The form of mapping described in Chapter 10.2 is rating modification mapping.

A more drastic alternative is *grammar transformation mapping*. In this approach, the mapping function alters the structure of the problem solving grammar by using the characteristics of the meta-level results to eliminate rules. This results in a corresponding elimination of operators. By eliminating the operators, the mapping function reduces the cost of problem solving by pruning search operators before they are considered.

For example, the results of approximate process may indicate that the solution to a vehicle tracking interpretation problem must be a vehicle track of type "A." As a consequence, the mapping function might eliminate all operators that are not on paths to an "A" interpretation. Chapter 13 and Appendix D.4 discuss this form of mapping in greater detail.

A third alternative is *explicit plan mapping*. Here, the mapping operator explicitly reorders the sequence of operators on the queue. This is in contrast to *implicit plan mapping* in which the mapping operator adjusts the *UPC* values associated with an operator and then allows the queue reordering to be conducted implicitly with the established rating function. We do not experiment with this form of mapping in this thesis.

10.4 Chapter Summary

This chapter introduces the IDP approach to representing a variety of sophisticated control mechanisms in a heuristic problem solver. The control mechanisms that are modeled include preconditions, goal processing, bounding functions, and approximate processing mechanisms.

CHAPTER 11

EXPERIMENTAL APPROACH WITH A HEURISTIC PROBLEM SOLVER

One of the primary goals of the work described in this thesis is to demonstrate that the IDP/*UPC* framework can be used to analytically determine the expected performance of a sophisticated problem solver in a complex domain. This section presents the organization and results of experiments designed to address this goal. Section 11.1 presents the problem domain used in the experiments. Section 11.2 describes the architecture of the problem solver used in these experiments. The results of experiments are detailed in Section 11.3.

11.1 The Experimental Problem Domain

The problem domain used in these experiments is based on the vehicle tracking domain described in previous chapters and summarized with the generational grammar shown in Fig. 11.1. Two general forms of this grammar are used in the experiments. Grammar VT1, which is equivalent to that shown in the figure, models a domain in which there are multiple independent ("I"), pattern ("P"), and ghost ("G") tracks. This grammar is used in experiment 1. Grammar VT2 is used in experiments 2 through 6 and it differs only slightly from VT1. Specifically, VT2 supports only single track "scenarios." This is accomplished by modifying the original grammar's rule 2 so that the ψ value of the first RHS is 0 and the value of the second RHS is 1. Thus, the grammar only creates single track phenomena.

This change is required to test the forms of pruning used in these experiments. The static and dynamic pruning operators used include an implicit assumption that there is only a single track to be interpreted. Consequently, without this change, the static and dynamic pruning functions would eliminate many paths from consideration that are actually correct paths.

Even with the modified grammar, scenarios can include multiple vehicle tracks. Pattern tracks consist of two vehicle tracks and Ghost tracks consist of a vehicle track and a "reflection" or "ghost" of that track.

The actual grammar used in the experiments is also modified so that it creates paths of length 6. In general, it is not feasible or realistic to model domains where phenomena are allowed to represent unrestricted time sequences. The amount of data in a realistic domain would be overwhelming. Instead, real-world domains must employ some form of time-slicing in which data from one given period is analyzed before proceeding to the next time period. For these experiments, the time period was chosen to be 6.

The original vehicle tracking grammar was based on a model of a sensed area with vehicles moving in and out of the area. The grammar would generate vehicle track data that would originate on a boundary of the sensed region and traverse some portion of the region. In these experiments, the grammar generates tracks that are then "centered" in the imaginary sensed region. The current interpretation model does not include a component representing sensor degradation as a function of distance from the center of the region or any similar representations that take into account a phenomenon's position within the sensed region, so this assumption has no noticeable effect on the experiments other than to simplify the implementation.

1.	$S_{[f]}$	$\rightarrow$ $Tracks_{[f]}$	p=1				
2.	$Tracks_{[f]}$	$\rightarrow$ $Tracks_{[f]}$ $Track_{[f]}$	p=0.1				
		$\rightarrow$ $Track_{[f]}$	p=0.9				
3.	$Track_{[f]}$	$\rightarrow$ I-Track1$_{[f]}$	p=0.25				
		$\rightarrow$ I-Track2$_{[f]}$	p=0.25				
		$\rightarrow$ P-Track1$_{[f]}$	p=0.10				
		$\rightarrow$ P-Track2$_{[f]}$	p=0.10				
		$\rightarrow$ G-Track1$_{[f]}$	p=0.15				
		$\rightarrow$ G-Track2$_{[f]}$	p=0.15				
4.	I-Track1$_{[f]}$	$\rightarrow$ I-Track1$_{[f,\,t+1,\,x+V+A,\,y+V+A]}$ $T1_{[f]}$	p=1				
5.	I-Track2$_{[f]}$	$\rightarrow$ I-Track2$_{[f,\,t+1,\,x+V+A,\,y+V+A]}$ $T2_{[f]}$	p=1				
6.	P-Track1$_{[f]}$	$\rightarrow$ P-Track1$_{[f,\,t+1,\,x+V+A,\,y+V+A]}$ $P\text{-}T1_{[f]}$	p=1				
7.	P-Track2$_{[f]}$	$\rightarrow$ P-Track2$_{[f,\,t+1,\,x+V+A,\,y+V+A]}$ $P\text{-}T2_{[f]}$	p=1				
8.	G-Track1$_{[f]}$	$\rightarrow$ G-Track1$_{[f,\,t+1,\,x+V+A,\,y+V+A]}$ $G\text{-}T1_{[f]}$	p=1				
9.	G-Track2$_{[f]}$	$\rightarrow$ G-Track2$_{[f,\,t+1,\,x+V+A,\,y+V+A]}$ $G\text{-}T2_{[f]}$	p=1				
10.	P-T1$_{[f]}$	$\rightarrow$ $T1_{[f,\,t,\,x+O,\,y+O]}$ $T2_{[f]}$	p=1	11.	P-T2$_{[f]}$	$\rightarrow$ $T2_{[f,\,t,\,x+O,\,y+O]}$ $T2_{[f]}$	p=1
12.	G-T1$_{[f]}$	$\rightarrow$ $GT1_{[f,\,t,\,x+O,\,y+O]}$ $T1_{[f]}$	p=1	13.	G-T2$_{[f]}$	$\rightarrow$ $GT2_{[f,\,t,\,x+O,\,y+O]}$ $T2_{[f]}$	p=1
14.	$T1_{[f]}$	$\rightarrow$ $V1_{[f]}$ $N_{[f]}$	p=1	15.	$T2_{[f]}$	$\rightarrow$ $V2_{[f]}$ $N_{[f]}$	p=1
16.	$GT1_{[f]}$	$\rightarrow$ $GV1_{[f]}$ $N_{[f]}$	p=1	17.	$GT2_{[f]}$	$\rightarrow$ $GV2_{[f]}$ $N_{[f]}$	p=1
18.	$N_{[f]}$	$\rightarrow$ $n_{[f]}$ $N_{[f]}$	p=0.1	19.	$V1_{[f]}$	$\rightarrow$ $G1_{[f]}$ $G3_{[f]}$ $G7_{[f]}$	p=0.4
		$\rightarrow$ $n_{[f]}$	p=0.25			$\rightarrow$ $G1_{[f]}$ $G3_{[f]}$	p=0.3
		$\rightarrow$ λ	p=0.65			$\rightarrow$ $G1_{[f]}$ $G7_{[f]}$	p=0.25
20.	$V2_{[f]}$	$\rightarrow$ $G3_{[f]}$ $G8_{[f]}$ $G12_{[f]}$	p=0.4			$\rightarrow$ λ	p=0.05
		$\rightarrow$ $G8_{[f]}$ $G12_{[f]}$	p=0.3	21.	$GV1_{[f]}$	$\rightarrow$ G-$G1_{[f]}$ G-$G3_{[f]}$ G-$G7_{[f]}$	p=0.2
		$\rightarrow$ $G3_{[f]}$ $G12_{[f]}$	p=0.25			$\rightarrow$ G-$G1_{[f]}$ G-$G3_{[f]}$	p=0.3
		$\rightarrow$ λ	p=0.05			$\rightarrow$ G-$G1_{[f]}$ G-$G7_{[f]}$	p=0.25
22.	$GV2_{[f]}$	$\rightarrow$ G-$G3_{[f]}$ G-$G8_{[f]}$ G-$G12_{[f]}$	p=0.4			$\rightarrow$ λ	p=0.05
		$\rightarrow$ G-$G8_{[f]}$ G-$G12_{[f]}$	p=0.3	23.	$G1_{[f]}$	$\rightarrow$ $S1_{[f]}$ $S2_{[f]}$	p=0.45
		$\rightarrow$ G-$G3_{[f]}$ G-$G12_{[f]}$	p=0.25			$\rightarrow$ $S1_{[f]}$ $S3_{[f]}$	p=0.1
		$\rightarrow$ λ	p=0.05			$\rightarrow$ $S1_{[f]}$ $S4_{[f]}$	p=0.1
24.	$G3_{[f]}$	$\rightarrow$ $S5_{[f]}$ $S7_{[f]}$	p=0.45			$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$	p=0.1
		$\rightarrow$ $S5_{[f]}$ $S6_{[f]}$	p=0.1			$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$ $S4_{[f]}$	p=0.1
		$\rightarrow$ $S6_{[f]}$ $S7_{[f]}$	p=0.1			$\rightarrow$ $S1_{[f]}$	p=0.05
		$\rightarrow$ $S4_{[f]}$ $S5_{[f]}$	p=0.1			$\rightarrow$ $S2_{[f]}$	p=0.05
		$\rightarrow$ $S7_{[f]}$ $S8_{[f]}$	p=0.1			$\rightarrow$ λ	p=0.05
		$\rightarrow$ $S5_{[f]}$	p=0.05	25.	$G7_{[f]}$	$\rightarrow$ $S11_{[f]}$ $S15_{[f]}$	p=0.55
		$\rightarrow$ $S7_{[f]}$	p=0.05			$\rightarrow$ $S11_{[f]}$ $S16_{[f]}$	p=0.43
		$\rightarrow$ λ	p=0.05			$\rightarrow$ λ	p=0.02
26.	$G8_{[f]}$	$\rightarrow$ $S13_{[f]}$ $S18_{[f]}$	p=0.55	27.	$G12_{[f]}$	$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$ $S17_{[f]}$	p=0.45
		$\rightarrow$ $S13_{[f]}$ $S17_{[f]}$	p=0.1			$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$	p=0.25
		$\rightarrow$ $S14_{[f]}$ $S18_{[f]}$	p=0.1			$\rightarrow$ $S7_{[f]}$ $S14_{[f]}$ $S18_{[f]}$	p=0.25
		$\rightarrow$ $S15_{[f]}$ $S17_{[f]}$	p=0.1			$\rightarrow$ λ	p=0.05
		$\rightarrow$ $S13_{[f]}$	p=0.05	29.	G-$G3_{[f]}$	$\rightarrow$ $S5_{[f]}$ $S7_{[f]}$	p=0.2
		$\rightarrow$ $S18_{[f]}$	p=0.05			$\rightarrow$ $S5_{[f]}$ $S6_{[f]}$	p=0.05
		$\rightarrow$ λ	p=0.05			$\rightarrow$ $S6_{[f]}$ $S7_{[f]}$	p=0.05
28.	G-$G1_{[f]}$	$\rightarrow$ $S1_{[f]}$ $S2_{[f]}$	p=0.2			$\rightarrow$ $S4_{[f]}$ $S5_{[f]}$	p=0.05
		$\rightarrow$ $S1_{[f]}$ $S3_{[f]}$	p=0.05			$\rightarrow$ $S7_{[f]}$ $S8_{[f]}$	p=0.05
		$\rightarrow$ $S1_{[f]}$ $S4_{[f]}$	p=0.05			$\rightarrow$ $S5_{[f]}$	p=0.2
		$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$	p=0.05			$\rightarrow$ $S7_{[f]}$	p=0.15
		$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$ $S4_{[f]}$	p=0.05			$\rightarrow$ λ	p=0.25
		$\rightarrow$ $S1_{[f]}$	p=0.2	31.	G-G8$_{[f]}$	$\rightarrow$ $S13_{[f]}$ $S18_{[f]}$	p=0.15
		$\rightarrow$ $S2_{[f]}$	p=0.2			$\rightarrow$ $S13_{[f]}$ $S17_{[f]}$	p=0.05
		$\rightarrow$ λ	p=0.2			$\rightarrow$ $S14_{[f]}$ $S18_{[f]}$	p=0.05
30.	G-$G7_{[f]}$	$\rightarrow$ $S11_{[f]}$ $S15_{[f]}$	p=0.30			$\rightarrow$ $S15_{[f]}$ $S17_{[f]}$	p=0.05
		$\rightarrow$ $S11_{[f]}$ $S16_{[f]}$	p=0.30			$\rightarrow$ $S13_{[f]}$	p=0.2
		$\rightarrow$ λ	p=0.40			$\rightarrow$ $S18_{[f]}$	p=0.25
32.	G-$G12_{[f]}$	$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$ $S17_{[f]}$	p=0.2			$\rightarrow$ λ	p=0.25
		$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$	p=0.2	35.	$n_{[f,\,t]}$	$\rightarrow$ $S1_{[f]}$	p=0.05
		$\rightarrow$ $S7_{[f]}$ $S14_{[f]}$ $S18_{[f]}$	p=0.25			$\rightarrow$ $S2_{[f]}$	p=0.05
		$\rightarrow$ λ	p=0.35	...	...	...	...
						$\rightarrow$ $S20_{[f]}$	p=0.05

Figure 11.1. Grammar Rules for a Vehicle Tracking Domain

11.2 The Experimental Problem Solving Architecture

The experimental problem solver used a blackboard control strategy based on six levels of abstraction: signal events, group events, vehicle events, vehicle tracks, pattern tracks, and solutions. The control architecture is based on that shown in Fig. 11.2. This architecture includes use of preconditions, pruning functions, and goal processing as described in Chapter 10.

The problem solver's termination criteria is a fully connected search space and a selection of a "correct" interpretation. As defined in Chapter 3, a search space is fully connected when the path from every state lead to either a final solution (i.e., a complete interpretation) or an explicit termination. Thus, a fully connected search space corresponds with an empty operator queue.

The decision regarding the "correct" interpretation is made by selecting the interpretation with the highest rating. This does not necessarily correspond to the domain event that generated the signal data. Thus, the problem solver does not generate an interpretation and halt. It generates all possible interpretations and selects the one with the highest rating.

Also, it is important to note that data is used exclusively in this system. Therefore, no single piece of data can be used in two different ways within the same interpretation. For example, if a pattern track involves two crossing tracks, there must be data to independently support each individual track. In scenarios that include multiple tracks, the problem solver cannot construct a solution in which multiple tracks share a single piece of data. Such solutions are invalid.

The operator evaluation function used is specified in Equation 11.2.1 and is based on the *rating* (i.e., credibility) assigned to a partial result that an operator is extending and its level on the blackboard. Ratings are determined by the semantic functions of the grammar and are based on the concept of *consistency*. As described in Chapter 3, the semantic functions define a partial result's credibility in a recursive manner. In the experimental problem solver, they are used in an inverse manner to calculate ratings.

Equation 11.2.1 $R(op_i(n_j)) = LEVEL(n_j) * CRED(n_j)$,

where $R(op_i)$ is the rating for the potential problem solving operator op_i applied to search state n_j, $LEVEL(n_j)$ is the level of the blackboard corresponding to state n_j, and $CRED(n_j)$ is n_j's credibility.

The concept of consistency represents the degree to which individual components of an interpretation are consistent with each other. For example, in a natural language understanding domain, consistency would represent the degree to which a verb phrase and a noun phrase "make sense." In this simulated vehicle tracking domain, we represent a variety of domain characteristics in the rating metric. These include energy level of the signal (which is represented as credibility), turning radius (vehicles are not expected to turn or decelerate sharply), and length of track (short tracks are considered anomalies).

The number of a partial result's level is used in the evaluation function to rate the problem solving actions that can be applied to the partial result. This has the effect of skewing the heuristic problem solving strategy to be more depth-first by increasing the ratings of partial results that are on "higher" levels of the blackboard.

The knowledge sources used by the problem solver correspond to the production rules of the domain grammar and are summarized in Tables 11.1, 11.2, and 11.3. These operators are, in large part, automatically constructed.

There are several modifications that are made to the set of operators that are not reflected in the domain grammar. These are required to correctly process partial interpretations at the

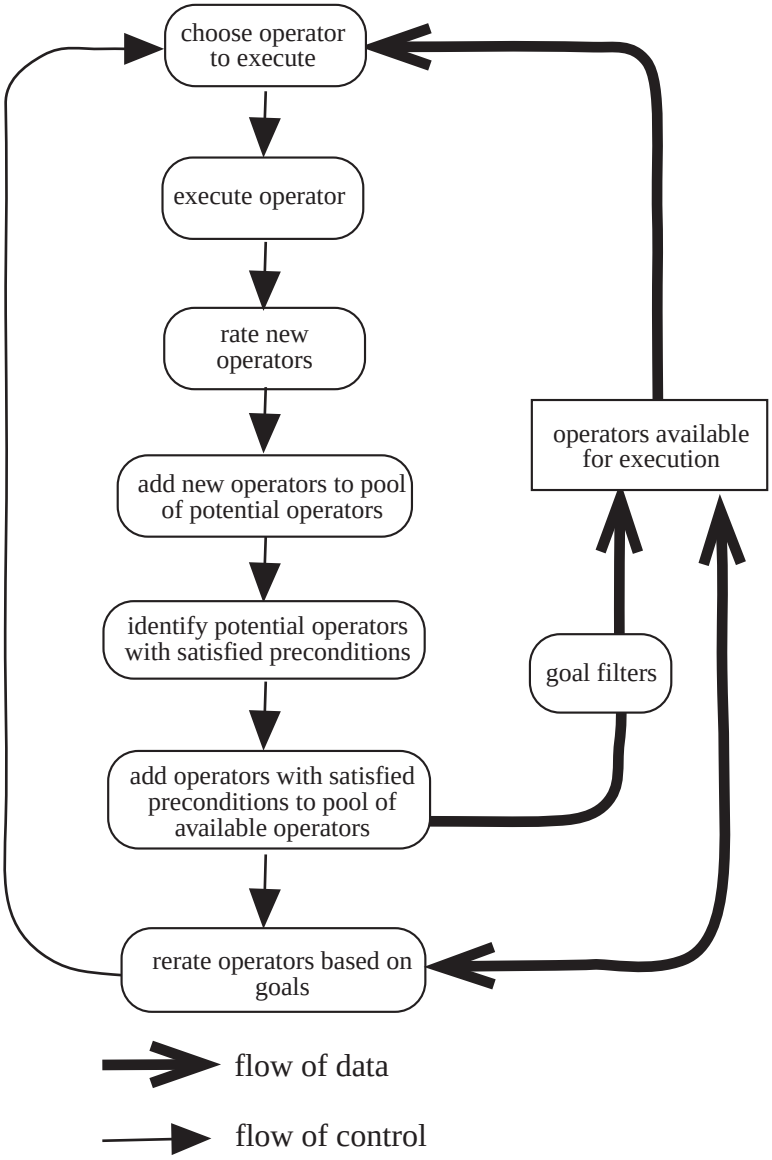


Figure 11.2. The Basic Control Cycle For the Experimental Problem Solver

Table 11.1. Summary of Track and Scenario Problem Solving Operators

IDP_i Rule	Operator	BB Level	Description
$\text{Track}_{[f]} \rightarrow \text{I-Track1}_{[f]}$	SC-IT1	Scenario	synthesis of I-Track1 scenarios
$\text{Track}_{[f]} \rightarrow \text{I-Track2}_{[f]}$	SC-IT2	Scenario	synthesis of I-Track2 scenarios
$\text{Track}_{[f]} \rightarrow \text{P-Track1}_{[f]}$	SC-PT1	Scenario	synthesis of P-Track1 scenarios
$\text{Track}_{[f]} \rightarrow \text{P-Track2}_{[f]}$	SC-PT2	Scenario	synthesis of P-Track2 scenarios
$\text{Track}_{[f]} \rightarrow \text{G-Track1}_{[f]}$	SC-GT1	Scenario	synthesis of P-Track1 scenarios
$\text{Track}_{[f]} \rightarrow \text{G-Track2}_{[f]}$	SC-GT2	Scenario	synthesis of P-Track2 scenarios
$\text{I-Track1}_{[f]} \rightarrow \text{I-Track1}_{[f,t+1,x+V+A,y+V+A]}\ \text{T1}_{[f]}$	EX-T11	Track	I-Track1 Forward Extension
$\text{I-Track1}_{[f]} \rightarrow \text{I-Track1}_{[f,tb,te,x+V+A,y+V+A]}\ \text{T1}_{[tb-1,f]}$	EX-T12	Track	I-Track1 Backward Extension
$\text{I-Track1}_{[f]} \rightarrow \text{T1}_{[t,f]}\ \text{T1}_{[t+1,f]}$	EX-T13	Track	I-Track1 Initialization
$\text{I-Track2}_{[f]} \rightarrow \text{I-Track2}_{[f,t+1,x+V+A,y+V+A]}\ \text{T2}_{[f]}$	EX-T21	Track	I-Track2 Forward Extension
$\text{I-Track2}_{[f]} \rightarrow \text{I-Track2}_{[f,tb,te,x+V+A,y+V+A]}\ \text{T2}_{[tb-1,f]}$	EX-T22	Track	I-Track2 Backward Extension
$\text{I-Track2}_{[f]} \rightarrow \text{T2}_{[t,f]}\ \text{T2}_{[t+1,f]}$	EX-T23	Track	I-Track2 Initialization
$\text{P-Track1}_{[f]} \rightarrow \text{P-Track1}_{[f,t+1,x+V+A,y+V+A]}\ \text{P-T1}_{[f]}$	EX-PT11	Track	P-Track1 Forward Extension
$\text{P-Track1}_{[f]} \rightarrow \text{P-Track1}_{[f,tb,te,x+V+A,y+V+A]}\ \text{P-T1}_{[tb-1,f]}$	EX-PT12	Track	P-Track1 Backward Extension
$\text{P-Track1}_{[f]} \rightarrow \text{P-T1}_{[t,f]}\ \text{P-T1}_{[t+1,f]}$	EX-PT13	Track	P-Track1 Initialization
$\text{P-Track2}_{[f]} \rightarrow \text{P-Track2}_{[f,t+1,x+V+A,y+V+A]}\ \text{P-T2}_{[f]}$	EX-PT21	Track	P-Track2 Forward Extension
$\text{P-Track2}_{[f]} \rightarrow \text{P-Track2}_{[f,t+1,x+V+A,y+V+A]}\ \text{P-T2}_{[f]}$	EX-PT22	Track	P-Track2 Backward Extension
$\text{P-Track2}_{[f]} \rightarrow \text{P-T2}_{[t,f]}\ \text{P-T2}_{[t+1,f]}$	EX-PT23	Track	P-Track2 Initialization
$\text{G-Track1}_{[f]} \rightarrow \text{G-Track1}_{[f,t+1,x+V+A,y+V+A]}\ \text{G-T1}_{[f]}$	EX-GT11	Track	G-Track1 Forward Extension
$\text{G-Track1}_{[f]} \rightarrow \text{G-Track1}_{[f,t+1,x+V+A,y+V+A]}\ \text{G-T1}_{[f]}$	EX-GT12	Track	G-Track1 Backward Extension
$\text{G-Track1}_{[f]} \rightarrow \text{G-T1}_{[t,f]}\ \text{G-T1}_{[t+1,f]}$	EX-GT13	Track	G-Track1 Initialization
$\text{G-Track2}_{[f]} \rightarrow \text{G-Track2}_{[f,t+1,x+V+A,y+V+A]}\ \text{G-T2}_{[f]}$	EX-GT21	Track	G-Track2 Forward Extension
$\text{G-Track2}_{[f]} \rightarrow \text{G-Track2}_{[f,t+1,x+V+A,y+V+A]}\ \text{G-T2}_{[f]}$	EX-GT22	Track	G-Track2 Backward Extension
$\text{G-Track2}_{[f]} \rightarrow \text{G-T2}_{[t,f]}\ \text{G-T2}_{[t+1,f]}$	EX-GT23	Track	G-Track2 Initialization

Abbreviations

ITn: Individual Track of class n
PTn: Pattern Track of class n
GTn: Ghost Track of class n

track levels. For production rules that generate individual vehicle track locations and that have the form:
$\text{a-Track}_{[f]} \rightarrow \text{a-Track}_{[f,tb,te,x+V+A,y+V+A]}\ \text{a}_{[te+1,f]}$,
where tb and te represent the begin and end times of the track, an additional problem solving operator is created to extend the track backward in time. Specifically, the following rule is added to the set of operators:
$\text{a-Track}_{[f]} \rightarrow \text{a-Track}_{[f,tb,te,x+V+A,y+V+A]}\ \text{a}_{[tb-1,f]}$.

In other words, the first production rule combines a vehicle location partial result, a, at time t with a track that has an end time of t-1. The second production rule combines a vehicle location partial result, a, at time t with a track that has a begin time of t+1. These rules are required to allow the problem solver to do island driving by extending a track either forward or backward in time. Note that these production rules allow the corresponding operator to start with either a track partial result and extend it with a vehicle location partial result, or vice-versa.

In addition, a variety of rules have been added to initiate track level partial results. Specifically, for each production rule of the form:
a-Track$_{[f]}$ $\rightarrow$ a-Track2$_{[f,tb,te,x+V+A,y+V+A]}$ a$_{[te+1,f]}$,
a new production rule is added that does not correspond to any rule in the domain grammar. This rule has the form:
a-Track$_{[f]}$ $\rightarrow$ a$_{[t,f]}$ a$_{[t+1,f]}$. This rule allows track partial results to be initially formed by combining two adjacent vehicle location partial results. This additional rule is necessary because the production rule used to generate track data is inadequate for creating track level interpretations. Specifically, it requires that a track level partial results already be present on the blackboard in order to create a track. The new rule defeats this conundrum by allowing the interpretation of a track level partial result from two adjacent vehicle level partial results. Without this additional rule, there would be no operator to create the original tracks that are then extended by the existing track rules.

As previously mentioned, in the experimental domain, scenarios are generated with a path length of six and this data is then "centered" in the problem solver's sensed region. This is not reflected in either the domain grammar or the problem solving operators. Thus, operators attempt to extend tracks past time 6 and before time 1. In the experiments described here, this has the effect of increasing the cost of problem solving. However, these costs could be avoided by creating a grammar that explicitly limited the track length syntactically. However, the results from such a reorganization would be more difficult to generalize.

Table 11.2. Summary of Track and Vehicle Location Problem Solving Operators

IDP_i Rule	Operator Name	BB Level	Description
P-$T1_{[f]} \rightarrow T1_{[f,t,x+O,y+O]}$ $T2_{[f]}$	S-PT1	Track Location	synthesis of P-Track1 locations
P-$T2_{[f]} \rightarrow T2_{[f,t,x+O,y+O]}$ $T2_{[f]}$	S-PT2	Track Location	synthesis of P-Track2 locations
G-$T1_{[f]} \rightarrow GT1_{[f,t,x+O,y+O]}$ $T1_{[f]}$	S-GT1	Track Location	synthesis of G-Track1 locations
G-$T2_{[f]} \rightarrow GT2_{[f,t,x+O,y+O]}$ $T2_{[f]}$	S-GT2	Track Location	synthesis of G-Track2 locations
$T1_{[f]} \rightarrow V1_{[f]}$ $N_{[f]}$	S-T1	Track Location	synthesis of I-Track1 locations
$T2_{[f]} \rightarrow V2_{[f]}$ $N_{[f]}$	S-T2	Vehicle Location	synthesis of I-Track2 locations
$GT1_{[f]} \rightarrow GV1_{[f]}$ $N_{[f]}$	S-GVL1	Vehicle Location	synthesis of G-Track1 locations
$GT2_{[f]} \rightarrow GV2_{[f]}$ $N_{[f]}$	S-GVL2	Vehicle Location	synthesis of G-Track2 locations
$N_{[f]} \rightarrow n_{[f]}$ $N_{[f]}$	S-N1	Vehicle Location	synthesis of noise
$N_{[f]} \rightarrow n_{[f]}$	S-N2	Vehicle Location	synthesis of noise
$V1_{[f]} \rightarrow G1_{[f]}$ $G3_{[f]}$ $G7_{[f]}$	S-V11	Vehicle Location	synthesis of V1
$V1_{[f]} \rightarrow G1_{[f]}$ $G3_{[f]}$	S-V12	Vehicle Location	synthesis of V1
$V1_{[f]} \rightarrow G1_{[f]}$ $G7_{[f]}$	S-V13	Vehicle Location	synthesis of V1
$V2_{[f]} \rightarrow G3_{[f]}$ $G8_{[f]}$ $G12_{[f]}$	S-V21	Vehicle Location	synthesis of V2
$V2_{[f]} \rightarrow G8_{[f]}$ $G12_{[f]}$	S-V22	Vehicle Location	synthesis of V2
$V2_{[f]} \rightarrow G3_{[f]}$ $G12_{[f]}$	S-V23	Vehicle Location	synthesis of V2
$GV1_{[f]} \rightarrow$ G-$G1_{[f]}$ G-$G3_{[f]}$ G-$G7_{[f]}$	S-GV11	Vehicle Location	synthesis of GV1
$GV1_{[f]} \rightarrow$ G-$G1_{[f]}$ G-$G3_{[f]}$	S-GV12	Vehicle Location	synthesis of GV1
$GV1_{[f]} \rightarrow$ G-$G1_{[f]}$ G-$G7_{[f]}$	S-GV13	Vehicle Location	synthesis of GV1
$GV2_{[f]} \rightarrow$ G-$G3_{[f]}$ G-$G8_{[f]}$ G-$G12_{[f]}$	S-GV21	Vehicle Location	synthesis of GV2
$GV2_{[f]} \rightarrow$ G-$G8_{[f]}$ G-$G12_{[f]}$	S-GV22	Vehicle Location	synthesis of GV2
$GV2_{[f]} \rightarrow$ G-$G3_{[f]}$ G-$G12_{[f]}$	S-GV23	Vehicle Location	synthesis of GV2

Abbreviations

ITn: Individual Track of class n
PTn: Pattern Track of class n
GTn: Ghost Track of class n
Tn: Individual Track Location of class n
Gn: Ghost Vehicle Location of class n
Vn: Vehicle of class n
GVn: Ghost Vehicle of class n

Table 11.3. Summary of Group Synthesis Problem Solving Operators

IDP_i Rule	Operator Name	BB Level	Description
$G1_{[f]} \rightarrow S1_{[f]}\ S2_{[f]}$	S-G11	Group	synthesis of G1
$G1_{[f]} \rightarrow S1_{[f]}\ S3_{[f]}$	S-G12	Group	synthesis of G1
$G1_{[f]} \rightarrow S1_{[f]}\ S4_{[f]}$	S-G13	Group	synthesis of G1
$G1_{[f]} \rightarrow S2_{[f]}\ S3_{[f]}$	S-G14	Group	synthesis of G1
$G1_{[f]} \rightarrow S2_{[f]}\ S3_{[f]}\ S4_{[f]}$	S-G15	Group	synthesis of G1
$G1_{[f]} \rightarrow S1_{[f]}$	S-G16	Group	synthesis of G1
$G1_{[f]} \rightarrow S2_{[f]}$	S-G17	Group	synthesis of G1
$G3_{[f]} \rightarrow S5_{[f]}\ S7_{[f]}$	S-G31	Group	synthesis of G3
$G3_{[f]} \rightarrow S5_{[f]}\ S6_{[f]}$	S-G32	Group	synthesis of G3
$G3_{[f]} \rightarrow S6_{[f]}\ S7_{[f]}$	S-G33	Group	synthesis of G3
$G3_{[f]} \rightarrow S4_{[f]}\ S5_{[f]}$	S-G34	Group	synthesis of G3
$G3_{[f]} \rightarrow S7_{[f]}\ S8_{[f]}$	S-G35	Group	synthesis of G3
$G3_{[f]} \rightarrow S5_{[f]}$	S-G36	Group	synthesis of G3
$G3_{[f]} \rightarrow S7_{[f]}$	S-G37	Group	synthesis of G3
$G7_{[f]} \rightarrow S11_{[f]}\ S15_{[f]}$	S-G71	Group	synthesis of G7
$G7_{[f]} \rightarrow S11_{[f]}\ S16_{[f]}$	S-G72	Group	synthesis of G7
$G8_{[f]} \rightarrow S13_{[f]}\ S18_{[f]}$	S-G81	Group	synthesis of G8
$G8_{[f]} \rightarrow S13_{[f]}\ S17_{[f]}$	S-G82	Group	synthesis of G8
$G8_{[f]} \rightarrow S14_{[f]}\ S18_{[f]}$	S-G83	Group	synthesis of G8
$G8_{[f]} \rightarrow S15_{[f]}\ S17_{[f]}$	S-G84	Group	synthesis of G8
$G8_{[f]} \rightarrow S13_{[f]}$	S-G85	Group	synthesis of G8
$G8_{[f]} \rightarrow S18_{[f]}$	S-G86	Group	synthesis of G8
$G12_{[f]} \rightarrow S6_{[f]}\ S14_{[f]}\ S17_{[f]}$	S-G121	Group	synthesis of G12
$G12_{[f]} \rightarrow S6_{[f]}\ S14_{[f]}$	S-G122	Group	synthesis of G12
$G12_{[f]} \rightarrow S7_{[f]}\ S14_{[f]}\ S18_{[f]}$	S-G123	Group	synthesis of G12

Abbreviations

Gn: Group Data of class n

Table 11.4. Summary of Ghost Group Level Synthesis Problem Solving Operators

IDP_i Rule	Operator Name	BB Level	Description
$G\text{-}G1_{[f]} \rightarrow S1_{[f]}\ S2_{[f]}$	S-GG11	Group	synthesis of G-G1
$G\text{-}G1_{[f]} \rightarrow S1_{[f]}\ S3_{[f]}$	S-GG12	Group	synthesis of G-G1
$G\text{-}G1_{[f]} \rightarrow S1_{[f]}\ S4_{[f]}$	S-GG13	Group	synthesis of G-G1
$G\text{-}G1_{[f]} \rightarrow S2_{[f]}\ S3_{[f]}$	S-GG14	Group	synthesis of G-G1
$G\text{-}G1_{[f]} \rightarrow S2_{[f]}\ S3_{[f]}\ S4_{[f]}$	S-GG15	Group	synthesis of G-G1
$G\text{-}G1_{[f]} \rightarrow S1_{[f]}$	S-GG16	Group	synthesis of G-G1
$G\text{-}G1_{[f]} \rightarrow S2_{[f]}$	S-GG17	Group	synthesis of G-G1
$G\text{-}G3_{[f]} \rightarrow S5_{[f]}\ S7_{[f]}$	S-GG31	Group	synthesis of G-G3
$G\text{-}G3_{[f]} \rightarrow S5_{[f]}\ S6_{[f]}$	S-GG32	Group	synthesis of G-G3
$G\text{-}G3_{[f]} \rightarrow S6_{[f]}\ S7_{[f]}$	S-GG33	Group	synthesis of G-G3
$G\text{-}G3_{[f]} \rightarrow S4_{[f]}\ S5_{[f]}$	S-GG34	Group	synthesis of G-G3
$G\text{-}G3_{[f]} \rightarrow S7_{[f]}\ S8_{[f]}$	S-GG35	Group	synthesis of G-G3
$G\text{-}G3_{[f]} \rightarrow S5_{[f]}$	S-GG36	Group	synthesis of G-G3
$G\text{-}G3_{[f]} \rightarrow S7_{[f]}$	S-GG37	Group	synthesis of G-G3
$G\text{-}G7_{[f]} \rightarrow S11_{[f]}\ S15_{[f]}$	S-GG71	Group	synthesis of G-G7
$G\text{-}G7_{[f]} \rightarrow S11_{[f]}\ S16_{[f]}$	S-GG72	Group	synthesis of G-G7
$G\text{-}G7_{[f]} \rightarrow S11_{[f]}\ S15_{[f]}$	S-GG73	Group	synthesis of G-G7
$G\text{-}G8_{[f]} \rightarrow S13_{[f]}\ S18_{[f]}$	S-GG81	Group	synthesis of G-G8
$G\text{-}G8_{[f]} \rightarrow S13_{[f]}\ S17_{[f]}$	S-GG82	Group	synthesis of G-G8
$G\text{-}G8_{[f]} \rightarrow S14_{[f]}\ S18_{[f]}$	S-GG83	Group	synthesis of G-G8
$G\text{-}G8_{[f]} \rightarrow S15_{[f]}\ S17_{[f]}$	S-GG84	Group	synthesis of G-G8
$G\text{-}G8_{[f]} \rightarrow S13_{[f]}$	S-GG85	Group	synthesis of G-G8
$G\text{-}G8_{[f]} \rightarrow S18_{[f]}$	S-GG86	Group	synthesis of G-G8
$G\text{-}G12_{[f]} \rightarrow S6_{[f]}\ S14_{[f]}\ S17_{[f]}$	S-GG121	Group	synthesis of G-G12
$G\text{-}G12_{[f]} \rightarrow S6_{[f]}\ S14_{[f]}$	S-GG122	Group	synthesis of G-G12
$G\text{-}G12_{[f]} \rightarrow S7_{[f]}\ S14_{[f]}\ S18_{[f]}$	S-GG123	Group	synthesis of G-G12

Abbreviations
GGn: Ghost Group Data of class n

The experiments described in the following sections demonstrate that the analysis tools accurately predict the effects of various sophisticated control mechanisms, preconditions, pruning functions, and goal processing. Each of these mechanisms are implemented as previously described.

11.2.1 Precondition Mechanisms

A new precondition operator is added for each existing production rule of the grammar as described in Chapter 10.1. Precondition operators are automatically given a rating of "MAX" so that they are executed before any other operators are executed, as shown in Fig. 11.2. The precondition for a rule: A→ B C D has the two part form: 1) $A^{op}_{[B,C,D]} \rightarrow$ B C D; 2) A→ $A^{op}_{[B,C,D]}$. When executed, the precondition determines if the data necessary to match the RHS of the production rule is present on the blackboard. If the data is present, the state $A^{op}_{[B,C,D]}$ is created and operators applicable to it are assigned ratings by the evaluation function. If the precondition determines that the data necessary to match the RHS of the rule is not present on the blackboard, no new states are created. This effectively prevents all further problem solving along the path.

In the IDP/*UPC* representation, preconditions are considered to be separable knowledge sources rather than components of the control mechanism. In fact, in the original implementation of preconditions in the Hearsay II speech understanding system, preconditions were treated as knowledge sources rather than components of the control mechanism. Later, they were always executed immediately after being created. As a result, in later presentations, preconditions were not considered to be separate knowledge sources [V.R.Lesser *et al.*, 1975]. Thus, the treatment of preconditions here is reasonable and, as stated above, the problem solver forces preconditions to be executed before any domain actions.

This specification of a precondition is risky in the sense that it requires the problem solver to be "symmetric" in the sense that every production rule must be applicable to every member of the rule's RHS. For example, if a B, C, or D is added to the blackboard, the precondition must be executed. Otherwise, the precondition could prune paths when they should not be pruned. For example, for the rule $A^{op}_{[B,C,D]} \rightarrow$ B C D, assume that the corresponding problem solving operator can only be applied to a state corresponding to "D." If the "D" is created before the "B" and/or the "C" the precondition will be created and executed and it will prune the paths from "D" that would have created the "A." Since the operator is not applicable to "B" or "C," the "A" will never be created when, in fact, it should. Furthermore, this pruning is purely syntactic and will eliminate many solution paths that should not be eliminated.

11.2.2 Pruning Mechanisms

Two forms of pruning operators are used in the experiments, static and dynamic. Both forms are represented and function as described in Chapters 4 and 10. In the experiments conducted here, pruning functions were added to all track-level states. When executed, both static and dynamic operators compare the "expected" credibility rating of complete interpretations on a path from a partial result to a threshold. If the probability that the complete interpretation's rating will exceed the threshold is less than 10%, the path is pruned as described in Chapter 4.

The difference between the two forms of pruning is in the manner in which the thresholds are established. Static pruning uses a threshold that is determined a priori. Dynamic pruning

uses a threshold that is determined during processing by dynamically choosing the highest rating for complete interpretations generated so far.

11.2.3 Goal Processing

The third sophisticated control mechanism used in the experiments is goal processing described in Chapter 10.2. This mechanism is used to support more depth-first, goal driven problem solving by increasing the ratings of lower-level partial results that are needed to extend higher-level partial results. In the experiments, goal processing is added at the track level. For track-level partial results of the form $a - Track1_{[tb,te]}$, where tb and te represent the track's begin and end times, operators of the following form are added:
Goal-$a_{[tb-1]}$ $\rightarrow$ a-$Track1_{[tb,te]}$
and Goal-$a_{[te+1]}$ $\rightarrow$ a-$Track1_{[tb,te]}$. In other words, a "goal" is created to generate data needed to extend the track forward and backward in time.

The rating and blackboard level of the goal is identical to the partial result from which it is generated. For each goal, there is a mapping operator of the form:
$\emptyset \rightarrow$ Goal-$a_{[t]}$.
When executed, this operator finds all the partial results that can support the generation of the partial results that are needed to extend the track. It then finds all of the operators that can be used to extend the partial-results to new partial results that can support the extension of the track and attempts to modify their ratings. Ratings are modified by first calculating a new rating for an operator based on the rating of the high-level goal. If this operator evaluation rating is higher than the existing evaluation of the operator, the existing rating is replaced with the higher rating. After the mapping operator processes existing data, it establishes a filter that examines all newly generated data and processes it as described above.

The specific rerating process used in the experimental problem solver is as follows:

1. Determine a *goal rating* that corresponds to the rating of the operator(s) that require one or more elements from a set of partial results represented by the goal.

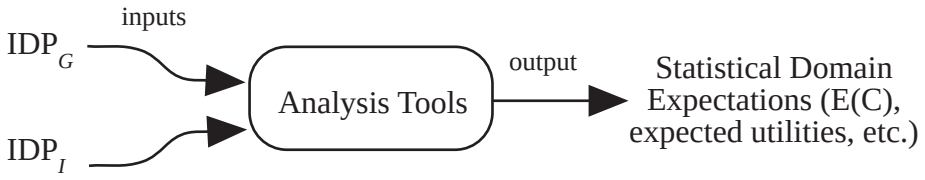
2. Find all intermediate states currently on the blackboard that are included in the component set of the goal state.

3. For each intermediate state, find all the associated operator instantiations currently on the queue that are components of paths to the goal.

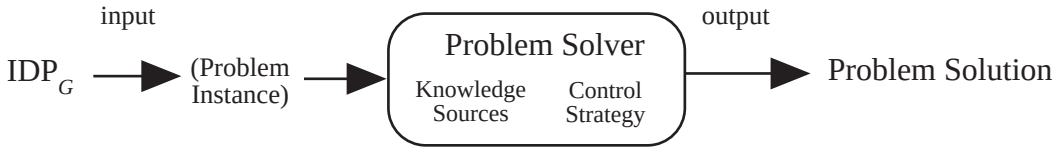
4. For each operator found in step 2, if the current rating is less than the *goal rating*, replace the operator's current rating with the *goal rating* and replace the operator in the queue.

5. Create a filter[1] for new operator instantiations. The filter functions as follows. For each new state, s, that is created, the filter determines if s is a member of the goal state's component set. If it is, the filter checks the ratings of all the operators instantiated based on s that are components of paths leading to the goal. If an operator's rating is lower than the *goal rating*, the operator's rating is replaced with the *goal rating*.

[1] Filters are implemented as a list of Lisp functions. For each new state created, each element of the filter function list is applied to each of the instantiated operators.



a. General organization of analysis system



b. General organization of problem solver

Figure 11.3. The Experimental Framework

11.3 Experimental Results

We have conducted a series of experiments based on the domain grammar shown in Fig. 11.1. The results of the experiments are shown in Table 11.5. The method used is illustrated in Fig. 11.3. As shown, the grammar was used to analyze the domain and predict the performance of a problem solver and to generate problem instances that were passed to the problem solver. By generating numerous problem instances and recording the performance of the problem solver, it was possible to statistically verify the validity of the analysis tools.

Four sets of experiments were conducted. In the first set, the problem solving operators corresponded to rules of the IDP$_g$ grammar shown in Fig. 11.1. In the second set of experiments, pruning operators were added to the grammar. The third set of experiments used goal processing operators and did not use pruning operators. The fourth set used both pruning and goal processing. The generational grammar was modified slightly in some of the experiments to facilitate analysis. This will be discussed in the following sections which give details of the experimental results.

11.4 Experiments with a Complex Grammar

Experiments 1 and 2 were conducted as base cases to which other experiments are compared. In addition, they verify that the analysis tools from Chapter 5 can be applied successfully in more complex, real-world domains. Note that in all of these experiments, scenarios are restricted to vehicle tracks of length six. This is necessary for computational purposes. It is extremely difficult to calculate and verify *UPC* values in situations where there is no limit to the length of the vehicle track. Though many techniques exist for calculating reasonable approximations for average path length in an unrestricted domain, these complicate the verification process unnecessarily as the six time frame restriction does not limit the applicability of the techniques to real-world domains, as discussed above.

Experiment 1 was conducted using a domain grammar that allows for multiple scenarios per run. Though multiple scenarios occur relatively infrequently, they result in a dramatic

increase in the cost of problem solving due to the interactions between partial results associated with different scenarios.

Experiment 2 uses a domain grammar that does not allow multiple scenarios in a single problem instance. This restriction greatly reduced the cost of problem solving, as expected. Again, the analysis tools accurately predicted the expected cost of problem solving.

In Table 11.5, the results of the heuristic problem solver are compared to a problem solver using an operator evaluation function based on *UPC* values. In the first two experiments, there is no difference between the expected results from heuristic search and *UPC* search. This is because the problem solving architecture does not use any form of pruning and is essentially doing an exhaustive search.

11.5 Experiments with Preconditions

In experiment 3, the problem solver was modified as described in Chapter 11.2 to incorporate the use of preconditions. As shown in Table 11.5, this resulted in a reduction in problem solving cost of over one-third. A quick review of the table indicates that this is the most effective technique used for reducing the cost of problem solving. Although it relies on the problem solver essentially being "symmetric," the use of preconditions does not preclude the problem solver from finding any correct answers.

In this experiment, the heuristic problem solver and the *UPC*-based problem solver again perform identically. Preconditions have several distinctive characteristics that lead to this result. First, the pruning is only one level deep. In other words, the problem solver does not need a sophisticated mechanism to estimate the long-range value of a pruning mechanism that is applied to search paths of length one. A heuristic function will compare favorably with the more complex potential calculation used by the *UPC* problem solver. In addition, the use of preconditions is *overwhelmingly* beneficial and is always the correct thing to do, especially when the cost is much less than the cost of domain processing. Therefore, if the heuristic always gives precondition operators the highest possible rating, it will always be taking the most efficient course of action.

11.6 Experiments with Pruning

Static and dynamic pruning operators are introduced in experiments 4 and 5 where they are incorporated into a problem solving architecture that also uses preconditions. As shown, the use of pruning operators further decreases the cost of problem solving by eliminating low-rated paths from consideration. Compared with static pruning, dynamic pruning is slightly more efficient at reducing the cost of problem solving and significantly better in terms of only pruning paths that do not lead to the highest rated interpretation.

Both pruning mechanisms were set to eliminate operators that only lead to complete interpretations with an expected rating that have less than a 10 % chance of exceeding the threshold. In the static pruning architecture, the threshold was set at 0.60. Thus, if the "U" value of a path has less than a 10% likelihood of exceeding 0.60, the path is pruned by setting its rating to 0. In the vehicle tracking problem domain, the expected rating of all scenarios is set to 0.7. During problem instance generation, this value is modified to represent the occurrence of various domain phenomena and then propagated to the low-level data by the generational grammar rules. Note that when the low-level data is interpreted, the final credibility may be significantly lower than the original credibility used to create the data. This is due to the effects of domain phenomena and is described more fully in Chapter 3.

Table 11.5. Results of Verification Experiments – Set 3

Exp	IDP_g	IDP_i	$E(C)$	Avg. C	$E(C)$*	Sig1	% Correct	E(% Correct)	Sig2
1	VT1	VT00	26,110	26,130	26,110	N	100	100	N
2	VT2	VT00	18,980	19,060	18,980	N	100	100	N
3	VT2	VT10	6,032	6,113	6,032	N	100	100	N
4	VT2	VT20	5,687	5,734	5,687	N	88	86	N
5	VT2	VT21	5,505	5,565	5,112	N	99	99	N
6	VT2	VT22	6,108	6,175	6,108	N	100	100	N
7	VT2	VT23	5,487	5,524	5,223	N	100	100	N
8	VT2	VT24	15,557	15,484	15,557	N	88	84	N
9	VT2	VT25	15,007	14,928	14,142	N	99	99	N
10	VT2	VT26	14,462	14,674	14,263	N	100	100	N
11	VT2	VT20	5,687	5,712	5,687	N	88	86	N
12	VT2	VT23	6,032	6,113	5,223	N	100	100	N

Abbreviations

Exp: Experiment

IDP_g: Description of IDP Domain Grammar used to generate problem instances.
VT1: complex vehicle tracking grammar (maximum/average path length of 6);
VT2: VT1 restricted to single scenarios (maximum/average path length of 6).

IDP_i: Description of IDP Interpretation Grammar specifying the problem solver.
VT00: complex vehicle tracking grammar (operator cost of 10);
VT10: VT00 with preconditions (cost 1);
VT20: VT10 with static pruning operators (cost 1);
VT21: VT10 with dynamic pruning operators (cost 1);
VT22: VT10 with goal processing at the I-, P-, and G-Track levels (cost 1);
VT23: VT21 and VT22 combined;
VT24: VT00 with static pruning operators (cost 1);
VT25: VT00 with dynamic pruning operators (cost 1);
VT26: VT25 with goal processing at the I-, P-, and G-Track levels (cost 1).

$E(C)$: Expected Cost of problem solving for given grammar and heuristic evaluation function.

$E(C)$*: Expected Cost of problem solving using exact UPC values.

Avg. C: actual average cost for 100 samples of 50 random problem instances each.

Sig1: Whether or not the difference between expected cost and the actual average cost was statistically significant, Y:yes; N:no.

% Correct : Percentage of correct answers found.

$E(\%Correct)$: Expected percentage of correct answers found by problem solver, where the correct answer is the interpretation with the highest credibility rating. Correct answers are not found in situations where paths to final states corresponding to the highest rated interpretation are pruned by bounding functions.

Sig2: Whether or not the difference between expected percentage of correct answers found and the actual percentage of correct answers found was statistically significant, Y:yes; N:no.

In these experiments, dynamic pruning performs more efficiently because it sets the threshold during problem solving after a complete interpretation is created and the average value for the threshold, 0.68, is higher than the one used in static pruning. More significantly, in certain instances, the dynamic threshold is set much higher than the static threshold, > 0.9, and there is a significant amount of pruning.

Furthermore, dynamic pruning is superior to static pruning in that it eliminates far fewer "correct" paths. In a typical problem instance, there is at least one component of a path that has a relatively low rating. In many instances, the static pruning operators eliminate paths from these components from consideration. In some cases, this does not matter because the symmetric nature of the problem domain compensates for the mistake. For example, a vehicle level partial result with a low rating may have all associated problem solving operators pruned. However, it is still possible to use this partial result in a track if the track extension operator is instantiated after the vehicle level data. In other cases, the static pruning operators can prune correct interpretations by eliminating critical elements.

Dynamic pruning is superior in this regard because it does not establish the pruning threshold until a full interpretation is formed. Although the first such interpretation is often not the best, the impact is the same because the low-level data is often shared with what turns out to be the "correct" or highest-rated interpretation. As a consequence, the data needed to create the highest-rated interpretation is generated before a threshold is established and is never subjected to pruning.

Note that although the table indicates that dynamic pruning found the correct solution 99% of the time, the actual results were much better. In actual runs, dynamic pruning rarely eliminated correct interpretations and the actual percentage of correct answers found is 99.8. The table truncates this value to draw attention to the fact that dynamic pruning does, in certain instances, eliminate the high-rated interpretations from consideration.

Also, it should be noted that for *most* problem instances, static pruning actually is more efficient. However, in instances where the complete interpretation found by dynamic pruning has a very high rating, > 0.9, it prunes significantly more low-level data than static pruning and thus reduces the overall expected cost of problem solving.

The use of static pruning has the same effect on both the heuristic and the *UPC* problem solver. This is because there is no "thought" involved in the pruning process. It does not matter what order the operators are applied, they will be subjected to the same pruning conditions.

The results of dynamic pruning, however, are quite different. In this experiment, the *UPC* problem solver performs much better than the heuristic problem solver. This is interesting to note because the heuristics used are intended to closely match the effects of using *UPC* values. The problem occurs because the heuristic problem solver is not as effective at generating high-quality results early that can be used to guide the pruning. Specifically, the heuristic problem solver's reliance on partial result ratings and on the weighting associated with a partial result's blackboard level to make control decisions was intended to have the effect of creating a more depth-first search that should lead to the generation of a complete interpretation more quickly than a breadth-first search. However, the heuristics fail when a component of a path has a relatively low rating. In these situations, the effort to construct a highly-rated complete interpretation stall and there is a delay in establishing a pruning threshold.

The *UPC* problem solver suffers from these effects to a lesser degree. The potential calculation functions as a means for seeing past the immediate implications of a low rating and it allows the problem solver to more accurately estimate the effects of processing the data to a level of the blackboard from which it can be incorporated into a more highly-rated result

that can be used to establish a pruning threshold. The specifics of this result are interesting. The potential calculation works most effectively in situations where the probability of creating a highly-rated complete interpretation using partial results with low ratings is likely from a syntactic perspective.

11.7 Experiments with Goal Processing

Goal processing is shown in conjunction with (experiment 7) and without (experiment 6) dynamic pruning. The results are shown in Table 11.6. Without the use of pruning mechanisms, goal processing simply increases the cost of problem solving. This is because the overhead of goal processing is incurred, but no benefits are derived.

When incorporated with dynamic pruning, goal processing has two effects. First, it significantly increases the efficiency of the pruning mechanisms to the point where they are comparable to the performance of the *UPC* problem solver. Second, they allow the problem solver to find the highest rated interpretation 100% of the time.

The use of goal processing reduces expected cost because it allows the problem solver to look past the immediate implications of a low rating on a partial result by replacing the low rating with a higher rating derived from the rating of a goal to extend a track. This causes the problem solver to more quickly generate highly-rated complete interpretations that can be used to prune other problem solving activities.

In addition, the use of goal processing eliminates situations where the problem solver prunes the data needed to construct the highest rated solution. When the pruning functions eliminate a path from consideration, they do not remove the associated operators from the set of potential activities. Instead, they set the rating of the operators to 0. The goal processing mechanisms search through the set of potential operators and modify the ratings of all applicable operators. As a consequence, the goal processing mechanisms restore previously pruned paths.

11.8 Experiments with Verifying Preconditions

Experiments 8, 9, and 10 demonstrate the advantages of preconditions. The results are shown in Table 11.7. The architecture used in these experiments included static pruning in experiment 8, dynamic pruning in experiment 9, and both goal processing and dynamic pruning in experiment 10. In each situation, the problem solver's performance failed to match the performance of corresponding architectures that used preconditions. Experiment 8 should be compared with experiment 4, 9 with 5, and 10 with 7, respectively.

11.9 Experiments with Alternate Evaluation Functions

Experiments 11 and 12 demonstrate the effects, shown in Table 11.8, of using an evaluation function that is not as appropriate for this domain. The results of experiment 4 should be compared with the results of experiment 11 and 7 should be compared with 12. The experiments were run using identical problem solving instances. This evaluation function used in 11 and 12 is more "breadth-first" and it is less efficient than the evaluation function used in 4 and 7. The evaluation function used in experiments 11 and 12 reversed the weightings of the blackboard levels. Thus a signal level operator had a blackboard coefficient of 6 and a scenario level operator had a blackboard coefficient of 1 in the evaluation function.

In experiments 4 and 11, there is no difference in the performance of the problem solvers. This is because, with a static pruning function, the evaluation function used has no impact.

Table 11.6. Comparison of Goal Processing Experiments

Exp	IDP_g	IDP_i	$E(C)$	Avg. C	$E(C)^*$	Sig1	% Correct	E(% Correct)	Sig2
6	VT2	VT22	6,108	6,175	6,108	N	100	100	N
7	VT2	VT23	5,487	5,524	5,223	N	100	100	N

Abbreviations

Exp: Experiment

IDP_g: Description of IDP Domain Grammar used to generate problem instances.
VT1: complex vehicle tracking grammar (maximum/average path length of 6);
VT2: VT1 restricted to single scenarios (maximum/average path length of 6).

IDP_i: Description of IDP Interpretation Grammar specifying the problem solver.
VT00: complex vehicle tracking grammar (operator cost of 10);
VT10: VT00 with preconditions (cost 1);
VT20: VT10 with static pruning operators (cost 1);
VT21: VT10 with dynamic pruning operators (cost 1);
VT22: VT10 with goal processing at the I-, P-, and G-Track levels (cost 1);
VT23: VT21 and VT22 combined;
VT24: VT00 with static pruning operators (cost 1);
VT25: VT00 with dynamic pruning operators (cost 1);
VT26: VT25 with goal processing at the I-, P-, and G-Track levels (cost 1).

$E(C)$: Expected Cost of problem solving for given grammar and heuristic evaluation function.

$E(C)^*$: Expected Cost of problem solving using exact UPC values.

Avg. C: actual average cost for 100 samples of 50 random problem instances each.

Sig1: Whether or not the difference between expected cost and the actual average cost was statistically significant, Y:yes; N:no.

% Correct : Percentage of correct answers found.

$E(\%Correct)$: Expected percentage of correct answers found by problem solver, where the correct answer is the interpretation with the highest credibility rating. Correct answers are not found in situations where paths to final states corresponding to the highest rated interpretation are pruned by bounding functions.

Sig2: Whether or not the difference between expected percentage of correct answers found and the actual percentage of correct answers found was statistically significant, Y:yes; N:no.

Table 11.7. Summary of Precondition Verification Experiments

Exp	IDP_g	IDP_i	$E(C)$	Avg. C	$E(C)^*$	Sig1	% Correct	E(% Correct)	Sig2
4	VT2	VT20	5,687	5,734	5,687	N	88	86	N
8	VT2	VT24	15,557	15,484	15,557	N	88	84	N
5	VT2	VT21	5,505	5,565	5,112	N	99	99	N
9	VT2	VT25	15,007	14,928	14,142	N	99	99	N
7	VT2	VT23	5,487	5,524	5,223	N	100	100	N
10	VT2	VT26	14,462	14,674	14,263	N	100	100	N

Abbreviations

Exp: Experiment

IDP_g: Description of IDP Domain Grammar used to generate problem instances.
VT1: complex vehicle tracking grammar (maximum/average path length of 6);
VT2: VT1 restricted to single scenarios (maximum/average path length of 6).

IDP_i: Description of IDP Interpretation Grammar specifying the problem solver.
VT00: complex vehicle tracking grammar (operator cost of 10);
VT10: VT00 with preconditions (cost 1);
VT20: VT10 with static pruning operators (cost 1);
VT21: VT10 with dynamic pruning operators (cost 1);
VT22: VT10 with goal processing at the I-, P-, and G-Track levels (cost 1);
VT23: VT21 and VT22 combined;
VT24: VT00 with static pruning operators (cost 1);
VT25: VT00 with dynamic pruning operators (cost 1);
VT26: VT25 with goal processing at the I-, P-, and G-Track levels (cost 1).

$E(C)$: Expected Cost of problem solving for given grammar and heuristic evaluation function.

$E(C)^*$: Expected Cost of problem solving using exact UPC values.

Avg. C: actual average cost for 100 samples of 50 random problem instances each.

Sig1: Whether or not the difference between expected cost and the actual average cost was statistically significant, Y:yes; N:no.

% Correct : Percentage of correct answers found.

$E(\%Correct)$: Expected percentage of correct answers found by problem solver, where the correct answer is the interpretation with the highest credibility rating. Correct answers are not found in situations where paths to final states corresponding to the highest rated interpretation are pruned by bounding functions.

Sig2: Whether or not the difference between expected percentage of correct answers found and the actual percentage of correct answers found was statistically significant, Y:yes; N:no.

Table 11.8. Comparison of Experiments Using Alternative Evaluation Functions

Exp	IDP_g	IDP_i	$E(C)$	Avg. C	$E(C)^*$	Sig1	% Correct	E(% Correct)	Sig2
4	VT2	VT20	5,687	5,734	5,687	N	88	86	N
11	VT2	VT20	5,687	5,712	5,687	N	88	86	N
7	VT2	VT23	5,487	5,524	5,223	N	100	100	N
12	VT2	VT23	6,032	6,113	5,223	N	100	100	N

Abbreviations

Exp: Experiment

IDP_g: Description of IDP Domain Grammar used to generate problem instances.
VT1: complex vehicle tracking grammar (maximum/average path length of 6);
VT2: VT1 restricted to single scenarios (maximum/average path length of 6).

IDP_i: Description of IDP Interpretation Grammar specifying the problem solver.
VT00: complex vehicle tracking grammar (operator cost of 10);
VT10: VT00 with preconditions (cost 1);
VT20: VT10 with static pruning operators (cost 1);
VT21: VT10 with dynamic pruning operators (cost 1);
VT22: VT10 with goal processing at the I-, P-, and G-Track levels (cost 1);
VT23: VT21 and VT22 combined;
VT24: VT00 with static pruning operators (cost 1);
VT25: VT00 with dynamic pruning operators (cost 1);
VT26: VT25 with goal processing at the I-, P-, and G-Track levels (cost 1).

$E(C)$: Expected Cost of problem solving for given grammar and heuristic evaluation function.

$E(C)^*$: Expected Cost of problem solving using exact UPC values.

Avg. C: actual average cost for 100 samples of 50 random problem instances each.

Sig1: Whether or not the difference between expected cost and the actual average cost was statistically significant, Y:yes; N:no.

% Correct : Percentage of correct answers found.

$E(\%Correct)$: Expected percentage of correct answers found by problem solver, where the correct answer is the interpretation with the highest credibility rating. Correct answers are not found in situations where paths to final states corresponding to the highest rated interpretation are pruned by bounding functions.

Sig2: Whether or not the difference between expected percentage of correct answers found and the actual percentage of correct answers found was statistically significant, Y:yes; N:no.

In each situation, the problem solver fully connects the search space and the order in which it does so is not of significance.

In experiments 7 and 12, the problem solver did significantly better using the evaluation function that proceeded depth-first with goal processing. In these experiments, the original evaluation function led to the generation of complete interpretations that were used to dynamically prune competing partial interpretations. Very little pruning occurred in experiment 12. For the most part, all processing at a lower level of the blackboard was completed before any processing at the next higher level was completed. As a result, by the time a complete interpretation was generated, there was very little activity left subject to pruning.

11.10 Chapter Summary

These experiments indicate that the analysis framework, which was originally based on the use of statistically optimal control decisions, can be extended to incorporate the use of

heuristic control functions if the analysis can be tied to statistical characteristics of the domain. These experiments further demonstrate that the framework can be used for analyzing not only heuristic evaluation control mechanisms, but also heuristic control decisions that incorporate other types of control functions such as pruning, goal processing, and preconditions.

CHAPTER 12

TOWARD GENERAL DESIGN PRINCIPLES AND THEORIES

Chapter 10.3 introduced definitions of approximate processing. In this section we present additional examples to illustrate the use of the IDP/*UPC* representation and to informally introduce some of the analysis techniques that have been developed. Chapter 12.1 introduces two design techniques for addressing some of issues associated with approximate processing. Chapter 12.2 presents additional approximate processing examples. These examples illustrate some of the problems that are encountered when designing approximate processing operators and control strategies. Chapter 12.3 presents examples of how the techniques from Chapter 12.1 can be used to analyze the structure of a domain and to design more effective approximation strategies. Section 12.4 introduces several basic analysis tools and techniques.

12.1 Analysis and Design Techniques for Approximate Processing

Two general analysis techniques will be illustrated. We will refer to these as *comparative cost analysis* and *constraint flow analysis*. Comparative cost analysis and constraint flow analysis are complementary tools. Constraint flow analysis is used to generate potential sophisticated control mechanisms and comparative cost analysis is used to conduct pairwise evaluations of alternative control mechanisms.

Using comparative cost analysis, the IDP/*UPC* framework can evaluate the relative worth of a sophisticated control mechanism by calculating the cost associated with generating and processing the partial interpretation associated with the sophisticated control mechanism and then comparing it with the cost of generating the partial interpretation the control mechanism is, essentially, replacing.

The power of comparative analysis is based on the context-free nature of the IDP_g and IDP_i grammars. As discussed in Chapters 4.7 and 10 of this thesis and in [Whitehair and Lesser, 1993], the implication of a context-free grammar is that the subproblems represented by different interpretation/generation subtrees interact in strictly defined ways. Thus, given a context-free representation of a problem, it is possible to analyze the characteristics of subproblems, such as expected cost, in well-defined ways. This implication is, at least superficially, counterintuitive in many real world domains and it may seem that restricting the applicability of the comparative analysis to context-free domains might limit its usefulness. However, as discussed in Section 4.7, a context-free grammar can be extended with our feature list convention to represent relationships that appear to be context-sensitive. The critical aspect of the feature list convention is that it does not alter the context-free nature of the IDP_g or IDP_i grammars. Consequently, the comparative analysis techniques described here can be applied within the IDP/*UPC* framework.

In contrast, if the representation of an interpretation domain requires a context-sensitive grammar or an unrestricted grammar, comparative analysis in the form used here will not

Interpretation Grammar *G'*

0. S → A | B	2. B → DEW
1. A → CD	4. E → jk
3. C → fg	6. W → xyz
5. D → hi	8. j → (signal data)
7. f → (signal data)	10. k → (signal data)
9. g → (signal data)	12. x → (signal data)
11. h → (signal data)	14. y → (signal data)
13. i → (signal data)	15. z → (signal data)

Figure 12.1. Interpretation Search Operators Shown as a Set of Production Rules

work. This is because, in evaluating the costs associated with a subtree of an interpretation graph, you must also factor in costs associated with other subtrees. Consequently, you cannot evaluate two alternative control mechanisms by comparing the characteristics of their associated subtrees. Instead, your analysis will have to take into account characteristics associated with other subtrees and this could become an arbitrarily complex and difficult computation.

Constraint flow analysis identifies potentially useful abstract states based on an analysis of relationships between base space states. This analysis relies on determining the constraint relationships that exist between states. As shown in Fig. 10.18, constraint can be thought of in two ways, top-down and sibling. Top-down constraints are propagated from a parent node in an interpretation graph to its children. Sibling constraints are propagated among the children of a single node. In terms of a grammar, the LHS of a rule passes constraints to the elements of its RHS, and the elements of the RHS constrain each other.

In synthesizing abstract states, a cost benefit ratio specifies which are the most effective abstract states to incorporate in problem solving. The costs are expressed in terms of the cost to generate the abstract state and subsequently map the implications back to the base space. The benefits are measured in terms of the degree to which the abstract state constrains or limits the ensuing search process.

Constraint flow analysis estimates the degree to which knowing the characteristics of a particular state in a search space constrains the generation of other states. It does this by identifying search states that can interact during the generation of an interpretation. (In the IDP formalism, these search states correspond to elements of the grammar.) Problem solving activities associated top-down constraints can be analyzed from two general directions, top-down, in which constraints from from parent nodes to their children and siblings, and bottom-up, in which constraints flow from children nodes to their siblings and then to their respective parents.

The process involves three steps. The first step is to create single-step connectivity matrices representing top-down and sibling constraints. In these matrices, the existence of a direct constraint relationship between two nodes of a graph (or elements of a grammar) is represented with a 1 in the matrix. In a grammar, a direct constraint relationship can be said to exist between the LHS of a rule and all the elements of its RHS and pairwise between all the elements of a RHS of some rule. This is shown in Figures 12.2 and 12.3, which show top-down and bottom-up constraints, and in Fig. 12.4, which shows sibling constraints. Note

	S	A	B	C	D	E	W	f	g	j	k	h	i	x	y	z
S		1	1													
A				1	1											
B					1	1	1									
C								1	1							
D												1	1			
E										1	1					
W														1	1	1
f																
g																
j																
k																
h																
i																
x																
y																
z																

Figure 12.2. Example of Single-Step, Top-Down Connectivity Matrix used in Constraint Flow Analysis

that the bottom-up constraint flow in Fig. 12.3 is simply the transpose of the single-step, top-down constraint flow in Fig 12.2. As will become clear later in this section, these two flows are used for distinct analysis purposes and for the sake of clarity, they will be represented separately rather than combined in a single matrix. However, to improve the readability of this document, the bottom-up matrix will not be shown in most situations unless this causes confusion. The grammar used in this example is shown in Fig. 12.1.

In Fig. 12.2, element A-C, where A represents the row and C the column, is 1 because A constrains C in a top-down fashion. Similarly, A-D, where A represents the row and d the column, is also a 1. The implication is that if you knew something about the characteristics of A, you would also know constraints on the characteristics of C and D. For example, if the known characteristics of A include a location, C and D might be forced to have the same location or a known function of the location of A. From a graphical perspective, top-down constraints are represented with directed arcs. Thus, the existence of a 1 in element A-C does not mean that element C-A is also 1.

Element C-D of the matrix in Fig. 12.4is also 1 and represents a sibling constraint. Again, the implication is that constraining D in some way might also constrain C. (More concrete examples will be given later in this chapter with a vehicle tracking problem.) Sibling constraints are bidirectional. Thus, if matrix element C-D is 1, then element D-C is also 1.

The second step is to take the transitive closure of the top-down, single-step connectivity matrix. This is shown in Fig. 12.5. The entries of this matrix represent all the components of the grammar that can be derived from specific element of the grammar in an arbitrary number of steps.

	S	A	B	C	D	E	W	f	g	j	k	h	i	x	y	z
S																
A	1															
B	1															
C		1														
D		1	1													
E			1													
W			1													
f				1												
g				1												
j						1										
k						1										
h					1											
i					1											
x							1									
y							1									
z							1									

Figure 12.3. Example of Single-Step, Bottom-Up Connectivity Matrix used in Constraint Flow Analysis

	S	A	B	C	D	E	W	f	g	j	k	h	i	x	y	z
S																
A																
B																
C					1											
D				1		1	1									
E					1		1									
W					1	1										
f									1							
g								1								
j											1					
k										1						
h													1			
i												1				
x															1	1
y														1		1
z														1	1	

Figure 12.4. Example of Single-Step Sibling Connectivity Matrix used in Constraint Flow Analysis

	S	A	B	C	D	E	W	f	g	j	k	h	i	x	y	z
S		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A				1	1			1	1			1	1			
B					1	1	1			1	1	1	1	1	1	1
C								1	1							
D												1	1			
E										1	1					
W														1	1	1
f																
g																
j																
k																
h																
i																
x																
y																
z																

Figure 12.5. Transitive Closure of Single-Step, Top-Down Connectivity Matrix

The third and final step is to take the transitive closure of the closed, single-step, top-down connectivity matrix with respect to the single-step, sibling matrix. This is accomplished by starting with the closed, single-step, top-down connectivity matrix and "or-ing" together rows i and j if element (i,j) = 1 in the single-step, sibling matrix. This is referred to as the *constraint connectivity matrix*. Elements that are 1 indicate that the grammar element represented by the row index can interact with grammar elements that are represented by the column index. Specifically, a "1" indicates that the two elements can be combined into a single interpretation, either partial or complete. This is an existential representation. There is no indication of how strong the relationship might be. The connectivity matrix after transitive closure is shown in Fig. 12.6.

In the constraint connectivity matrix, A is shown to constrain all of the elements that can be derived from it. C constrains not only the elements that can be derived from itself, but also the elements that can be derived from D. This is because C constrains D with a sibling constraint, and D subsequently constrains all the elements that can be derived from it. Consequently, by transitive closure of D's single-step top-down constraint relationships, C constrains all the elements that can be derived from D.

It is interesting to note that even though A constrains D, it does not constrain D's siblings, E and W. From an algorithmic perspective, this is because the only sibling constraints that are reflected in the connectivity matrix are direct relationships. In other words, the connectivity matrix does not reflect a transitive closure of all sibling relationships. Thus, A is not related to E and W. From an intuitive perspective, A does not constrain E and W because it cannot be combined into a single solution with either of them. These grammar elements never interact and their local characteristics are never combined. Consequently, they do not constrain each other in any way.

	S	A	B	C	D	E	W	f	g	j	k	h	i	x	y	z
S		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
A				1	1			1	1			1	1			
B					1	1	1			1	1	1	1	1	1	1
C					1			1	1			1	1			
D				1		1	1	1	1	1	1	1	1	1	1	1
E					1		1			1	1	1	1	1	1	1
W					1	1				1	1	1	1	1	1	1
f									1							
g								1								
j											1					
k										1						
h													1			
i												1				
x															1	1
y														1		1
z														1	1	

Figure 12.6. Example of Constraint Connectivity Matrix used in Constraint Flow Analysis

It is also worth noting that D is more connected than any of its siblings. This can be determined by summing the entries in row D of the constraint connectivity matrix. This means that a constraint on D will, more than likely, have a greater limiting effect on search than a constraint on any of its siblings. This is a significant observation because it can be used in the design of meta-operators and problem solving strategies.

For example, problem solving can be viewed as the production and application of constraint. As discussed by a number of researchers, including Berliner [Berliner, 1979] and Carver [III, 1990], search can be thought of in terms of either a "find best" or a "disprove rest" strategy. In Berliner's B* algorithm, the problem solver can either prove that one search path leads to a solution that is better than all others, or it can prove that other search paths lead to solutions that are worse than a specific path. Carver's differential diagnosis is similar in nature to a "disprove rest" strategy, but it is more sophisticated and uses intelligence about the structure of the search space to plan an efficient strategy for eliminating competing solution paths by resolving the uncertainty associated with the path. (The relationship between IDP/*UPC* analysis tools and Carver's work is discussed further in Chapter 12.4.)

The constraint connectivity matrix can be used to construct efficient strategies that are similar to "disprove rest" or differential diagnosis. For example, as shown in Fig. 12.6, the constraint connectivity matrix can be used as a guide to determine which elements of a search space might be most useful for eliminating competing solutions. In the case of element D, we can see that knowledge about its characteristics will have more of a constraining influence than, for example, knowledge about the characteristics of C, E, or W. Understanding this search space structure would allow a problem solver to focus its efforts on first constraining D, then propagating these constraints to other solution paths.

The following sections will illustrate how these analysis tools can be used in the design of approximation techniques. The next section will discuss several approximation techniques that can be used to reduce problem solving cost. It will also illustrate how these mechanisms can *fail* to reduce problem solving cost. The subsequent section, Chapter 12.3, illustrates how flow-analysis and comparative analysis can be used to address these issues and design more effective approximations.

12.2 Simple Approximate Processing Examples

Figure 12.7 shows two grammars that will be used in a series of analysis examples. Figure 12.7.a shows a simple grammar that is extended with the level hopping operators shown in Fig. 12.7.b. This grammar will be referred to as the Level Hopping Grammar. Similarly, Fig. 12.7.c shows a base grammar that is extended in Fig. 12.7.d with abstract operators that eliminate corroborating support. This grammar will be referred to as the ECS Grammar. Though similar, these grammars have subtle differences that will be used to illustrate several important issues.

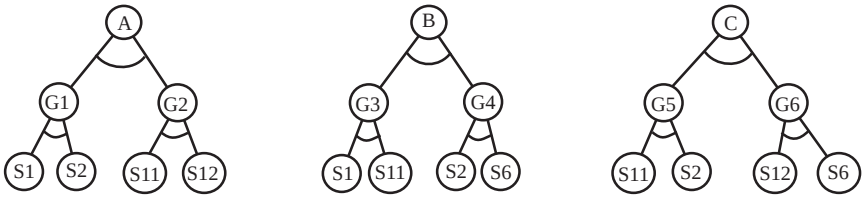
Figure 12.8 illustrates the use of comparative analysis of the ECS Grammar. The top half of the figure depicts the results of interpretation using the base space operators. The initial data appears in 12.8.a, 12.8.b shows the results of intermediate processing, and 12.8.c shows the full search space explored using the base space operators.

In contrast, the bottom half of the figure shows the results of problem solving using the ECS Grammar. Again, the initial data appears in 12.8.d, 12.8.e shows the results of intermediate abstract processing that generates the meta state A^{ecs}, and 12.8.f shows the full search space explored using the extended grammar with abstract operators.

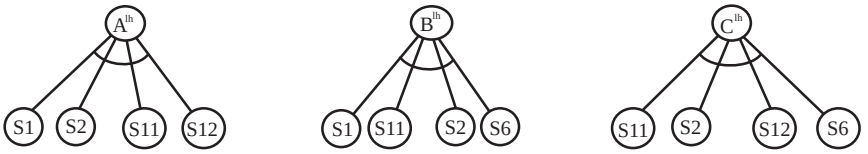
No precondition operators or goal processing operators are used in this example and, as a result, it is a simple example of comparative analysis. The two costs that are to be compared are the cost of fully connecting the search space shown in 12.8.c and the cost of generating and processing the abstract state A^{ecs}. Assuming the cost of all operator applications is 10, the cost of connecting the base space in 12.8.c is 150. This is calculated using the method described in Chapter 5.1. In general, this method determines cost by summing the cost associated with each state. The cost of each state is equal to the sum of the cost of applying all relevant operators to the state. In this example, the cost of applying all relevant operators is the product of a state's out degree and 10. i.e., each out arc from a state represents the application of an operator and the cost of each operator is 10. Although not shown in the figure, the out degree of G3 and G5 is 1.

In contrast, the cost of problem solving using the ECS grammar with abstract states is the sum of the cost of generating A^{ecs} and the cost of processing A^{ecs}. The cost of processing A^{ecs} is equal to the cost of generating all the elements of the *component set* of the base space state, A, that corresponds to the abstract state, A^{ecs}. The definition of a component set appears in Chapter 4 and is repeated in Definition 12.2.1.

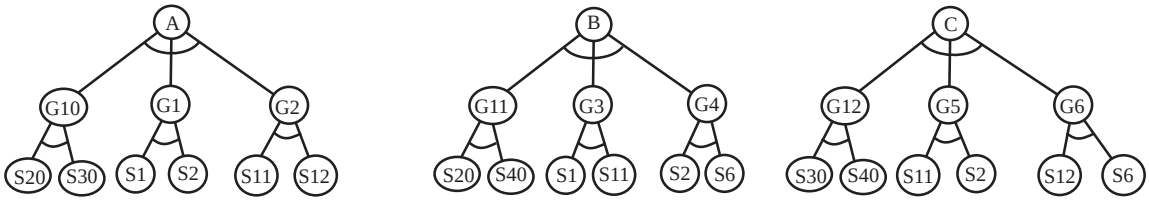
Definition 12.2.1 *Component Set (CS) - The component set of a state, s_n, includes all the states that lie on paths from the signal data to s_n. In terms of an interpretation grammar, IDP_i, the component set of a grammar element, E, includes all the grammar elements that can be derived from E, inclusive.*



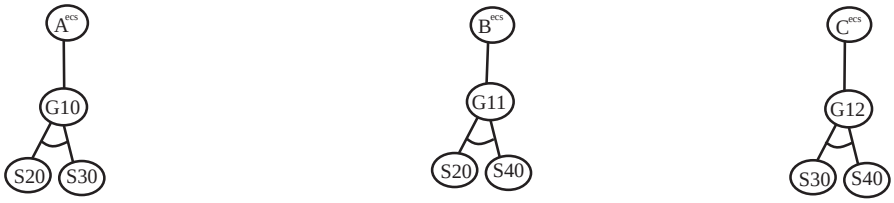
a. graphical representation of grammar defining base space operators



b. graphical representation ofgrammar extensions defining level-hopping operators

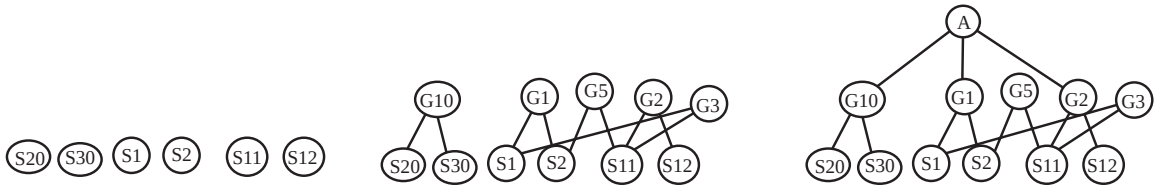


c. graphical representation of grammar defining base space operators



d. Graphical representation of grammar extensions defining operators that eliminate corroborating support

Figure 12.7. Level Hopping and ECS Example Grammars

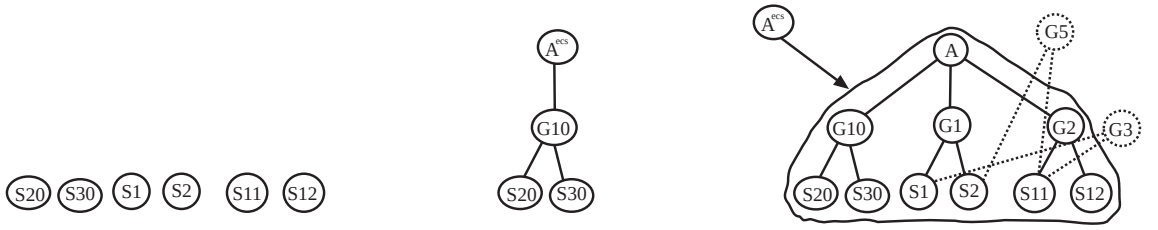


a. initial data

b. results of intermediate level processing

c. full search space explored during interpretation

Summary of problem solving activity without approximation



d. initial data

e. result of approximate processing that eliminates corroborating support (ecs)

f. result of mapping abstract ecs results back to the base space

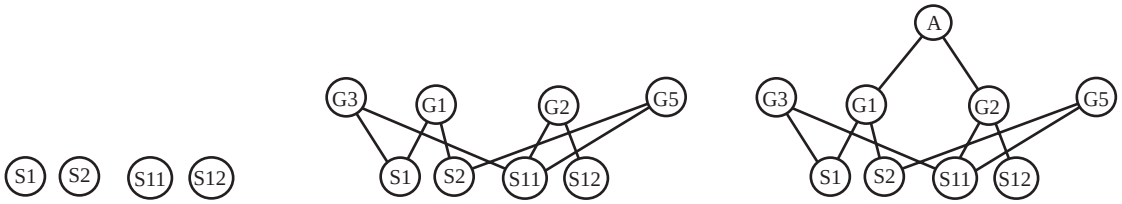
Summary of problem solving activity with approximation

Figure 12.8. Comparative Analysis Example Using the ECS Grammar

Let the cost of generating A^{ecs} be the cost of generating the intermediate state G10 plus the cost of the abstract ECS operator. For now we will assume that the cost of the ECS operator is 10. Thus, the cost of generating A^{ecs} is 30. The cost of processing A^{ecs} is the sum of the cost of mapping A^{ecs} back to the base space and the cost of connecting the enclosed search space shown in 12.8.f. The arrow from A^{ecs} represents mapping A^{ecs} back to the base space. This operator, when executed, functions by setting the rating of all intermediate operators that do not lead to the interpretation of an A to 0. This effectively prunes the generation of the states G5 and G3. Assuming the cost of mapping is 10 and the cost of connecting the enclosed search space is 70, the full cost of problem solving using the ECS abstraction is 110. (Note that the cost of connecting the enclosed search space in 12.8.f includes the cost of generating G10 and this cost is only included once in the total.) Consequently, in this example, the use of abstract operators for eliminating corroborating support reduce the cost of problem solving from 150 to 110.

Figure 12.9 illustrates the use of comparative analysis of the Level Hopping Grammar. The top half of the figure depicts the results of interpretation using the base space operators. The initial data appears in 12.9.a, 12.9.b shows the results of intermediate processing, and 12.9.c shows the full search space explored using the base space operators.

In contrast, the bottom half of the figure shows the results of problem solving using the Level Hopping Grammar. Again, the initial data appears in 12.9.d, 12.9.e shows the results of intermediate abstract processing that generates the meta state A^{lh}, and 12.9.f shows the full search space explored using the extended grammar with abstract operators.

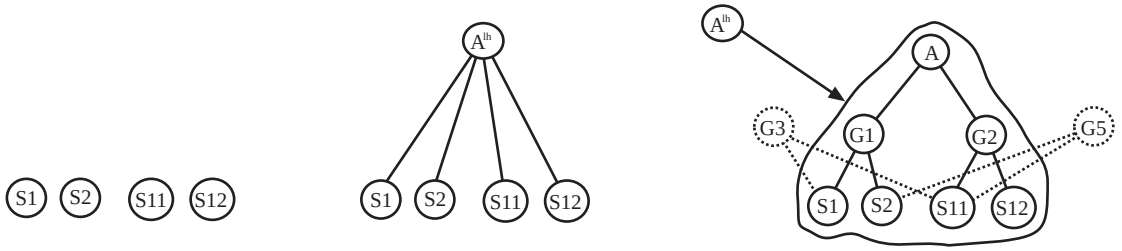


a. initial data

b. results of intermediate level processing

c. full search space explored during interpretation

Summary of problem solving activity without approximation



d. initial data

e. result of approximate level hopping

f. result of mapping abstract level hopping results back to the base space

Summary of problem solving activity with approximation

Figure 12.9. Comparative Analysis Example Using the Level Hopping Grammar

Assuming the cost of all operator applications is 10, the cost of connecting the base space in 12.9.c is 120.

In contrast, the cost of problem solving using the extended Level Hopping Grammar is the sum of the cost of generating A^{lh} and the cost of processing A^{lh}. Let the cost of generating A^{lh} be 10, the cost of applying a single instance of the level hopping grammar. (In this example and in the experiments that follow, level hopping is treated as a clustering or aggregating abstraction and its associated costs are computed as described in Chapter 13.1.) The cost of processing A^{lh} is the sum of the cost of mapping A^{lh} back to the base space (10) and the cost of connecting the enclosed search space shown in 12.9.f. As before, the mapping operator, when executed, functions by setting the rating of all intermediate operators that do not lead to the interpretation of an A to 0. This effectively prunes the generation of the states G5 and G3. Assuming the cost of mapping is 10 and the cost of connecting the enclosed search space is 60, the full cost of problem solving using the ECS abstraction is 80.

With both the ECS Grammar and the Level Hopping Grammar, comparative analysis indicates that the use of abstract operators is beneficial. In the case of the ECS Grammar, the cost of connecting the base space without using abstract states is 150. With the abstract states, the cost is 110. Similarly, in the case of the Level Hopping Grammar, the costs are 120 versus 80. In both situations, the analysis to determine whether or not the use of abstractions is beneficial was a direct comparison of the processing that occurred with and without the abstractions. No other computations or analysis was needed.

P.1.1.	T	→ T1 A
P.1.2.	T	→ T2 B
P.1.3.	T	→ T3 C
P.2.	T1	→ A A
P.3.	T2	→ B B
P.4.	T3	→ C C
P.5.	A	→ G1 G2
P.6.	B	→ G3 G4
P.7.	C	→ G5 G6
P.8.1.	G1	→ S1 S2
P.8.2.	G1	→ S1 S2 S6
P.9.	G2	→ S11 S12
P.10.1.	G3	→ S1 S11
P.10.2.	G3	→ S1 S11 S12
P.11.	G4	→ S2 S6
P.12.1.	G5	→ S2 S11
P.12.2.	G5	→ S1 S2 S11
P.13.	G6	→ S6 S12

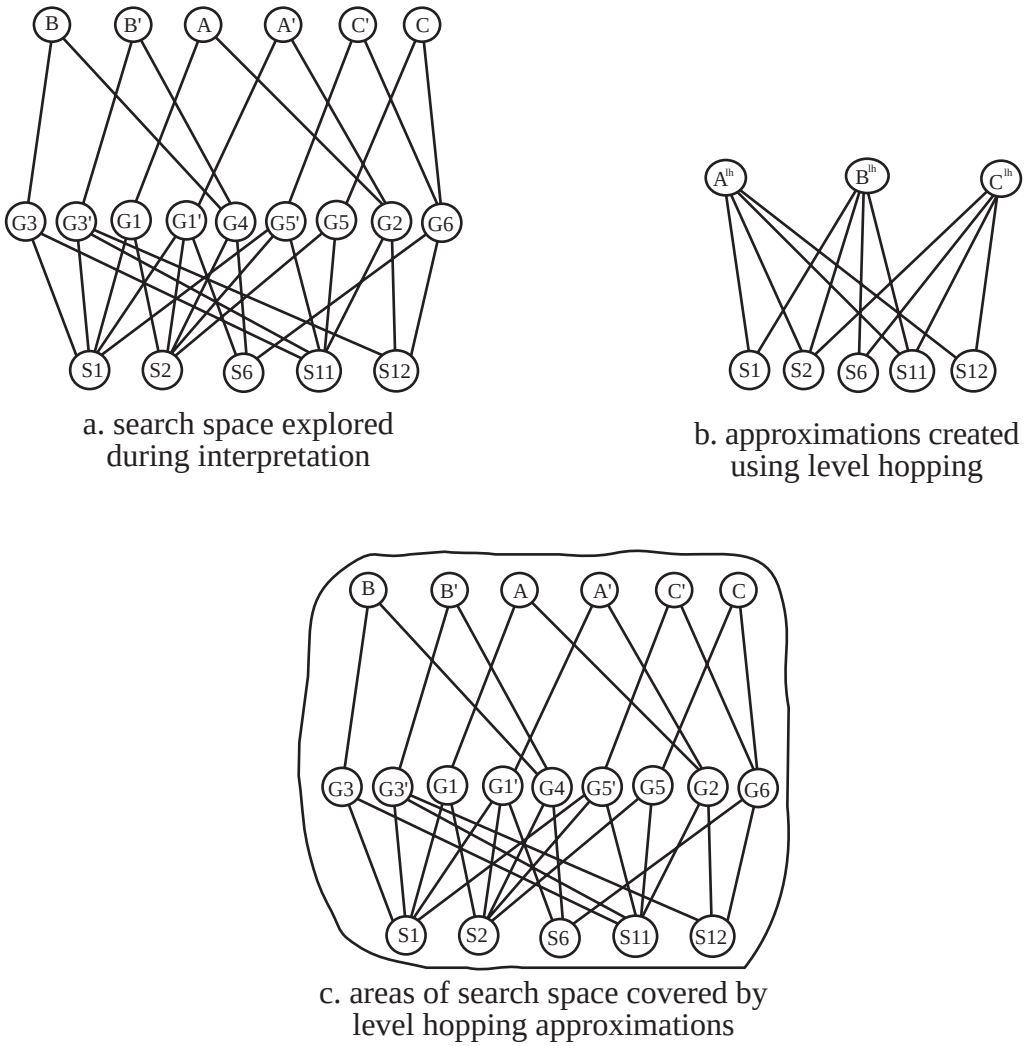
Figure 12.10. Full Grammar VTG-1 for Tracking Vehicles Through Multiple Time-Locations

12.3 Extended Approximate Processing Examples

In contrast to the two examples of the use of abstract processing from Chapter 12.2, consider the grammar shown in Fig. 12.10. This is a simple vehicle tracking grammar which we will call VTG-1. An example of processing with VTG-1 is shown in Fig. 12.11. This example is interesting because, as shown in the figure, level hopping does not appear to be effective because it does not prune any states. In fact, use of level hopping in this example actually seems to increase the cost of problem solving[1].

In Fig. 12.11.a, the full set of the search space explored during exhaustive processing of the given inputs for one time period is shown. The results of level hopping using the same input data is shown in Fig. 12.11.b. Figure 12.11.c shows the area of the search space that will be explored after the results of level hopping are mapped back to the base space. Quick inspection will reveal that the areas of the search space covered in 12.11.a and 12.11.b are identical. Therefore, the cost of search in the base space will be the same with or without level hopping, but the overall cost of problem solving will be increased by the cost of level hopping and mapping the results of level hopping back to the base space. If the approximate

[1]In general, the use of meta-operators will reduce the cost of problem solving in two ways, as described in [Decker *et al.*, 1990]. One is by eliminating certain search paths from consideration. This reduces the cost of problem solving because the costs of executing the operators required to explore that area are not incurred. Meta-operators can also generate information or otherwise constrain problem solving so that, even though no areas of the search space are eliminated from consideration, the cost of the operators used to expand the space are reduced. In other words, a meta-operator may not eliminate a problem solving activity, but it might significantly reduce its cost. For the sake of clarity, the discussions in this section will focus on cost reductions that involve eliminating areas of the search space from consideration.



a. search space explored during interpretation

b. approximations created using level hopping

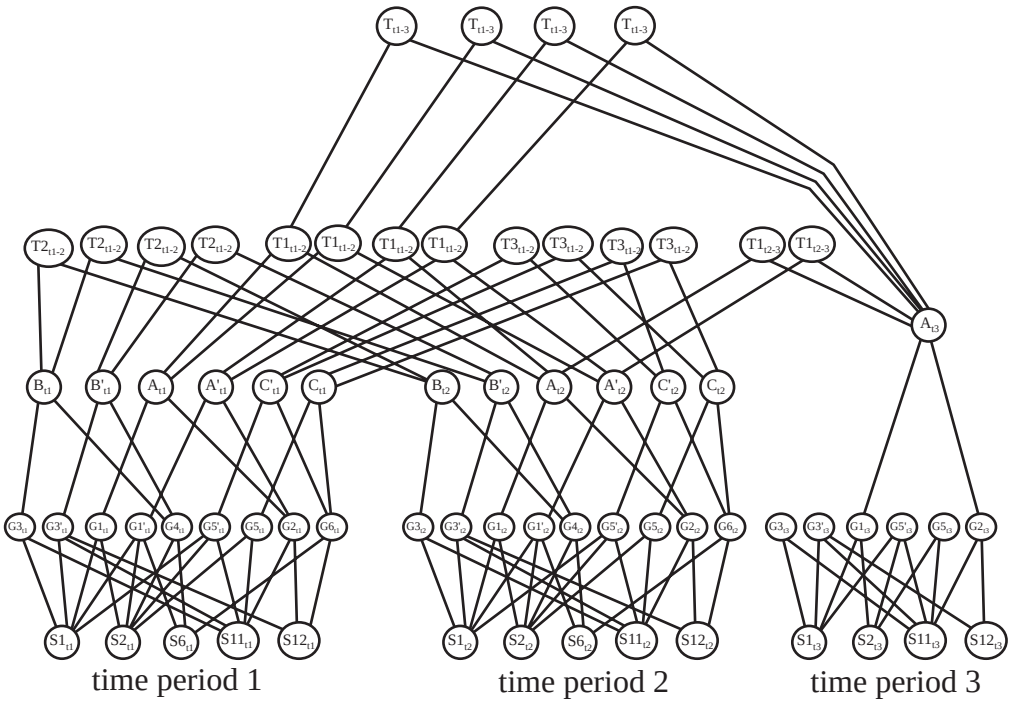
c. areas of search space covered by level hopping approximations

Figure 12.11. Example Problem Scenario With Level Hopping in VTG-1

processing used were based on eliminating corroborating support instead of level hopping, the results would be similar.

Figure 12.12 shows a fully expanded search space for a complete VTG-1 interpretation problem. There are three sets of signal data, each corresponding to a different time period. The results from the first two time periods are ambiguous. The signal data could correspond to any of the events A, B, and C. In fact, in each case, it is possible to generate ambiguous interpretations of each individual event. Thus, there are two ways to interpret an event A in time period 1, there are two ways to interpret an event B, etc.

The use of approximations based on level hopping or eliminating corroborating support at the A, B, and C event class level are not useful for reducing the cost of problem solving in this domain. However, it is not clear whether or not there is *some* form of approximation that could be used to reduce the cost of problem solving. We have developed methods based on the use of constraint flow analysis to address this issue.



fully expanded search space

Figure 12.12. Track Interpretation Example

The methodology we have developed is still in a formative phase. As mentioned throughout this thesis, the IDP/*UPC* framework is intended to form the basis of theories and principles for designing sophisticated control strategies that are based on the use of abstractions and approximations. Our current methodology is based on the *principle of differentiation*. This principle holds that, in order for an approximation to be useful, it must differentiate portions of the search space that can be eliminated from consideration and it must do so more efficiently than base-space processing.

Examining Fig. 12.12, we see that the ambiguities are differentiated only at the full track level. In fact, for the given data, there are four complete interpretations that ultimately must be differentiated by their credibilities. i.e., the "correct" interpretation is the one with the highest credibility. This is typical of most interpretation tasks. There are usually several (or even many) interpretations that will explain the observed data and the problem solver's task is to find the most credible one.

The ambiguity of this domain grammar is reflected in the constraint connectivity matrices, shown in Figures 12.13, the single-step, top-down connectivity matrix, 12.14, the single-step, sibling connectivity matrix, 12.15, the transitive closure of the single-step matrix, and in 12.16, the full constraint connectivity matrix.

In analyzing the flow of constraint in this domain, first notice that in Fig. 12.15, the transitive closure of top-down connectivity, the rows for grammar elements T, T1, T2, T3, A, B, and C are all similar through their connection to signal groups S1 - S12. This indicates that all of these grammar elements constrain signal groups S1 - S12. This is a reflection of the

	T	T1	T2	T3	A	B	C	G1	G2	G3	G4	G5	G6	S1	S2	S6	S11	S12
T		1	1	1	1	1	1											
T1					1													
T2						1												
T3							1											
A								1	1									
B										1	1							
C												1	1					
G1														1	1	1		
G2																	1	1
G3														1			1	1
G4															1	1		
G5														1	1		1	1
G6																1		1
S1																		
S2																		
S6																		
S11																		
S12																		

Figure 12.13. Single-Step, Top-Down Connectivity Matrix for VTG-1

	T	T1	T2	T3	A	B	C	G1	G2	G3	G4	G5	G6	S1	S2	S6	S11	S12
T																		
T1					1													
T2						1												
T3							1											
A					1													
B						1												
C							1											
G1									1									
G2								1										
G3											1							
G4										1								
G5													1					
G6												1						
S1															1	1	1	1
S2														1		1	1	
S6														1	1			1
S11														1	1			1
S12														1		1	1	

Figure 12.14. Single-step, Sibling Connectivity Matrix for VTG-1

	T	T1	T2	T3	A	B	C	G1	G2	G3	G4	G5	G6	S1	S2	S6	S11	S12
T		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
T1					1			1	1					1	1	1	1	1
T2						1				1	1			1	1	1	1	1
T3							1					1	1	1	1	1	1	1
A								1	1					1	1	1	1	1
B										1	1			1	1	1	1	1
C												1	1	1	1	1	1	1
G1														1	1	1		
G2																	1	1
G3														1			1	1
G4															1	1		
G5														1	1		1	1
G6																1		1
S1																		
S2																		
S6																		
S11																		
S12																		

Figure 12.15. Transitive Closure of Single-Step, Top-Down Connectivity Matrix for VTG-1

ambiguity of the grammar. It indicates that, from a purely syntactic perspective, these elements cannot be differentiated on an aggregate basis. In other words, the aggregated data is the same for all of these grammar elements and the only way to differentiate them is to actually interpret the data and compare credibility values. (Note that this illustrates the classic difference between interpretation tasks and classification tasks. Interpretation tasks require significant processing to disambiguate the potential answers.)

In contrast is a grammar in which some interpretations require a signal level event and some do not. In this situation, it would be possible to construct an abstraction that would search for this element in the aggregated data and, if it did not appear, eliminate from consideration all possible interpretations that required the signal level event.

Examining the entries for grammar elements G1 - G6, we see that there are significant differences at the group level. Specifically, the entries in each row corresponding to each group element's interaction with signal data is unique. This is significant because unique signal data row entries often indicate grammar structures that allow a problem solver to quickly differentiate areas of the search space that can be eliminated from consideration. This is discussed at greater length in Chapter 12.4. Unfortunately, the group level data is only one level removed from the signal data. Consequently, there is little to gain from constructing meta-operators at this level of the grammar. Such operators would essentially be the equivalent of existing problem solving operators.

Although there are no rows with unique entries corresponding to the signal data, it is also clear that each row is unique with respect to group level data. Unfortunately, as seen in the example in Fig. 12.12, the group level data is ambiguous. Even though the individual group level events are capable of differentiating potential solutions, their occurrence is ambiguous with each other. For example, the existence of a G2 group would seem to indicate that the

	T	T1	T2	T3	A	B	C	G1	G2	G3	G4	G5	G6	S1	S2	S6	S11	S12
T		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
T1					1			1	1					1	1	1	1	1
T2						1				1	1			1	1	1	1	1
T3							1					1	1	1	1	1	1	1
A					1			1	1					1	1	1	1	1
B						1				1	1			1	1	1	1	1
C							1					1	1	1	1	1	1	1
G1									1					1	1	1	1	1
G2								1						1	1	1	1	1
G3											1			1	1	1	1	1
G4										1				1	1	1	1	1
G5													1	1	1	1	1	1
G6												1		1	1	1	1	1
S1															1	1	1	1
S2														1		1	1	
S6														1	1			1
S11														1	1			1
S12														1		1	1	

Figure 12.16. Constraint Connectivity Matrix for Extended Grammar

interpretation must include an A. However, the data that generated the G2 might also generate a G3, which would indicate that the interpretation must include a B, or the data might generate a G5, indicating that the interpretation must include a C. These relationships are shown in the transitive closure of the top-down constraint matrix, Fig. 12.15.

Given that there are no other rows of the matrix that contain unique entries, it is necessary to try an alternative approach to finding an appropriate abstraction level. This alternative is required since the purely syntactic approach has failed. The alternative is to choose an abstraction level that encompasses enough of the search space to generate *semantic constraints* that can be used to differentiate the search space. The concept of a "semantic" approximation or abstraction indicates that the projection space defined by the approximation will contain operators that will process meta-states and generate new meta-states. Thus, the operators defined by $OP_{(\omega_a,\omega_a)}$, where a is an abstraction level, can be thought of as representing semantic approximations or abstractions. It would also be appropriate to think of the operators in $OP_{(\Omega,\omega_a)}$ as primarily syntactic approximations or abstractions. In other words, the operators that define projection spaces are typically approximations or abstractions of base-space grammar rules that alter the syntactic structure of the rules. The operators in the projection space, on the other hand, typically preserve the structure of the base-space rules and include simplified or approximated knowledge.

Referring to the connectivity matrix in Fig. 12.16, we see that the T level of the grammar constrains all the other elements of the grammar and, consequently, the entire search space. Consequently, T is a good candidate for the basis of an abstraction space that will incorporate semantic processing elements.

An alternative would be to construct an abstraction space based on T1, T2, or T3. Each of these encompasses a significant area of the search space and could form the basis for semantic

AP.1.	$AT_{c1 \cap c2}$	$\rightarrow AT1_{c1}\ VLC_{c2}$
AP.2.	$AT1_{c1 \cap c2}$	$\rightarrow VLC_{c1}\ VLC_{c2}$
C.1.	$VLC_{A \cup B \cup C}$	$\rightarrow A^{lh} \ldots B^{lh} \ldots C^{lh}$
LH.1.	A^{lh}	$\rightarrow$ S1 S2 S6 S11 S12
LH.2.	B^{lh}	$\rightarrow$ S1 S2 S6 S11 S12
LH.3.	C^{lh}	$\rightarrow$ S1 S2 S6 S11 S12

Figure 12.17. Approximations Used to Extend VTG-1

approximations. However, there are several problems this would present. First, this approach would require the specification of three distinct abstraction spaces, one corresponding to each of T1, T2, and T3. Second, these elements suffer from the same problem as the group level events. Specifically, a given set of data could lead to the generation of a T1, a T2, and a T3. Therefore, a given set of data could result in abstract processing occurring in three different projection spaces, none of which would be able to meaningfully differentiate the search space.

This illustrates an important issue. In order for certain uses of abstraction spaces to be effective, the abstract solution that gets mapped back to the base-space must be unique. In the examples we present in this thesis, the abstraction spaces are used to prune areas of the search space that are not consistent with an abstract solution determined in a projection space. If there are multiple projection spaces, each mapping its solution back to the base-space, this approach will not work. Each of the distinct projection space solutions might be mutually exclusive and the result would be the elimination of all data from consideration. For example, if T1, T2, and T3 projection spaces were formed, mapping solutions back to the base-space might be implemented so that the result of mapping T1's abstract solution would eliminate from consideration all solution paths except those that lead to T1, and so forth.

Figure 12.18 shows the operators that are used to extend VTG-1. The approximate operators that are used in subsequent experiments have the same form as those shown here. The rules LH.1, LH.2, and LH.3 are level hopping operators that generate an abstraction space. Rule C.1 is a clustering operator that combines multiple vehicle class events from the level hopping abstraction space and projects them to a new abstraction space. In this rule, the notation ... indicates that *all* results of the types shown, i.e., A^{lh}, B^{lh}, and C^{lh}, are combined with an aggregating operation that acts like a logical "or." Thus, if there is no data associated with one or two of the results, for example, if there is no data corresponding to an A^{lh} or B^{lh}, the operator will still successfully cluster any C^{lh} results, Furthermore, if there are multiple instances of any of the elements, they will all be aggregated into a single meta-level result. In the figure, the subscripts c1 and c2 indicate a set of event classes. These sets are generated by the clustering operator as shown in the figure. Rules AP.1 and AP.2 mirror the corresponding rules from the base-space grammar. As indicated in the figure, each combines two elements and takes the intersection of their event class sets.

The use of these grammar extensions is illustrated in Fig. 12.18 and 12.19. Figure 12.18 shows how the extensions would be applied to the example problem. As shown, the signal data generates abstract states in a level hopping projection space. For each of the distinct sets of signal data (sets of signal data correspond to the time period in which the events occurred), distinct states are created in the level hopping space. For time periods 1 and 2, three abstract states

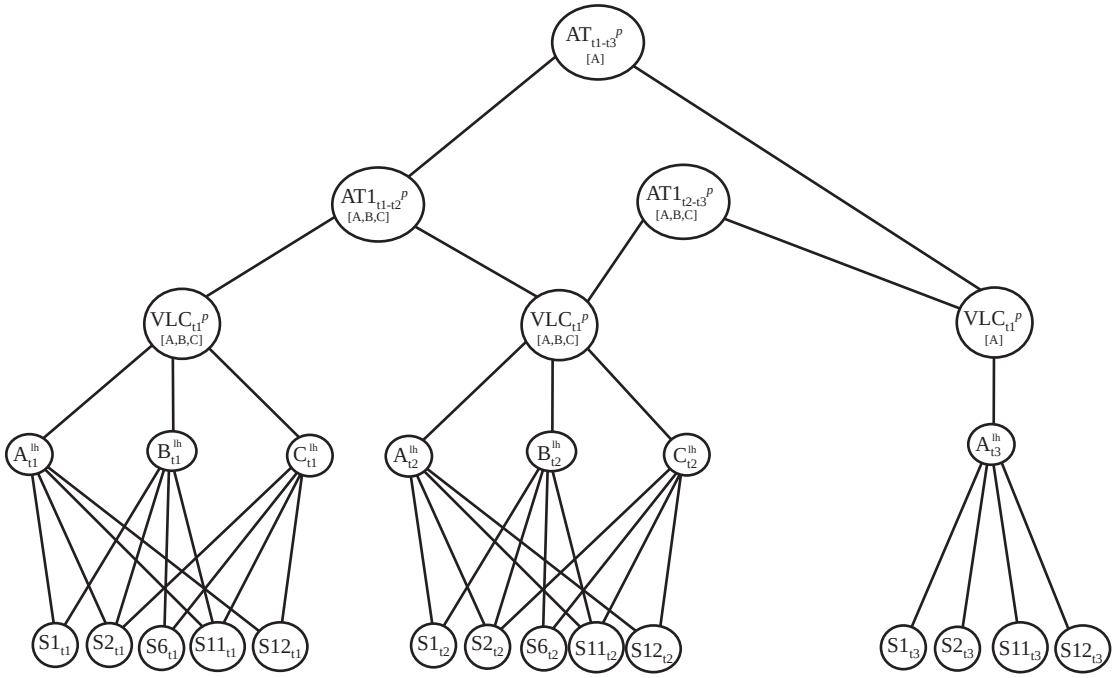


Figure 12.18. Track Level Abstraction Example

corresponding to vehicle level events A, B, and C are created in the level hopping abstraction space. For time period 3, only one abstract state is generate corresponding to an A vehicle level event class.

The level hopping results are then clustered into a new projection space. This space contains the states labeled VLC, for "vehicle location clusters." Each of these states has a subscript indicating the set of vehicle level event classes that are clustered together in the state.

The abstract states generated by clustering are processed in the same projection space by operators AP.2 and AP.1. These operators correspond to operators P.1 and P.2 from the base space. AP.2 combines two VLC elements into an approximate "T1" partial track. The event class set of the new state is the intersection of the sets from the component clusters. AP.1 combines a partial track with a VLC element, again taking the intersection of the component event class sets. The result is an abstract solution with a single event class, A.

Figure 12.19 illustrates the effects of mapping the results of problem solving in an abstract space back to the base-space. As shown, all search paths that lead to solutions for event classes other than A are eliminated. Still, the solution is ambiguous. There are four possible solutions that must be differentiated by their credibilities. However, the reduction is problem solving cost is substantial. The cost to fully expand the base search space is 1,320. There are 132 required search operations, each at a cost of 10. Using VTG-1 with the extensions for approximate processing, the cost of problem solving is only 580. This includes a cost of generating the abstract solution, 210, the cost of mapping the solution back to the search space, 10, and the cost of generating the four full interpretations, 360. For such a simple example, this is a very significant savings.

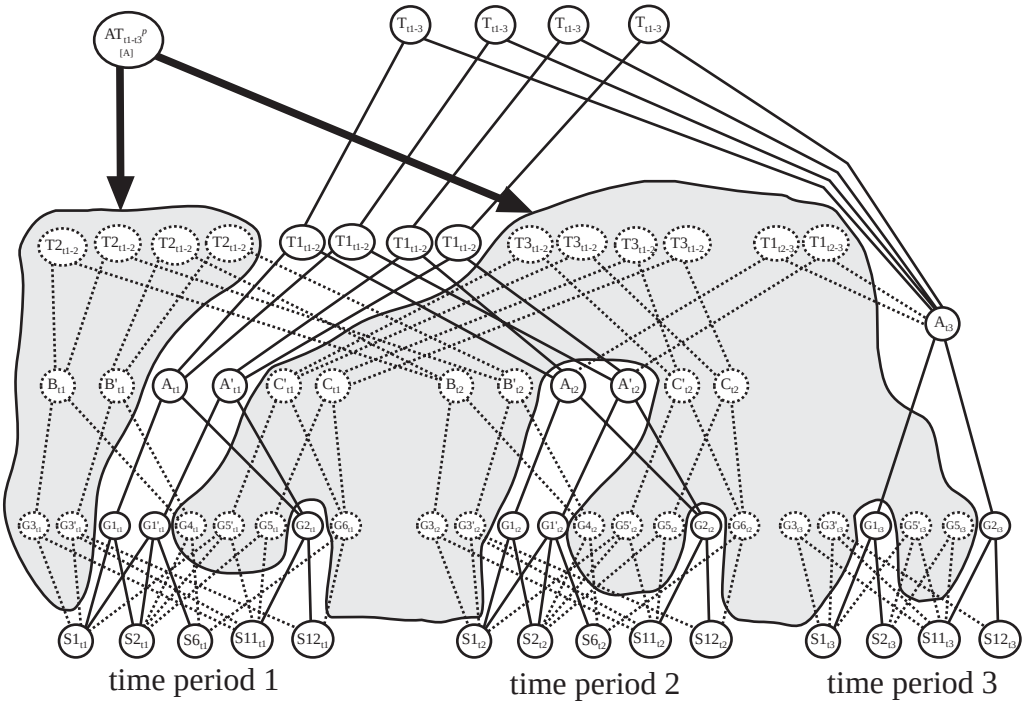


Figure 12.19. Track Processing Example

12.4 Basic Analysis Tools and Techniques

Given a formal IDP$_g$ model, it is possible to analytically characterize properties of the domain for use in explanation, prediction, and design of problem solver performance. In Chapter 5 we present analytical techniques for calculating a variety of different domain properties including expected cost of problem solving, expected frequencies of domain events, expected costs of individual interpretation search paths, the expected cost associated with incorrect search paths, ambiguity relationships, a quantified potential value for meta-operators, and more. Two specific domain structures are the concept of *marker* and *differentiator*. In the remainder of this section, we demonstrate the IDP$_g$ formalism by defining these two domain structures and we discuss how they can be used in dynamic control strategies and in the design of problem solver architectures.

The definitions of markers and differentiators are relative to the concept of a *solution nonterminal*, or *SNT*, described previously. For a given SNT, an intermediate result (i.e., a terminal or a nonterminal different from the SNT) is a *strong marker* if it is always (or, from a statistical perspective, almost always) *implied* by the SNT. Information about markers can be used to predict the occurrence and nature of low-level events and, thus, can be used to support top-down, model-based processing. Furthermore, analysis of the component structure can be used to determine the degree to which an intermediate result *differentiates* an SNT. Differentiation refers to the extent to which an intermediate result is exclusively associated with an SNT. The differentiation relationship is the inverse of the marker relationship and it can be used to support bottom-up, island-driving processing.

1.	$S_{[f]}$	$\rightarrow$ $Tracks_{[f]}$	p=1				
2.	$Tracks_{[f]}$	$\rightarrow$ $Tracks_{[f]}$ $Track_{[f]}$	p=0.1				
		$\rightarrow$ $Track_{[f]}$	p=0.9				
3.	$Track_{[f]}$	$\rightarrow$ I-Track1$_{[f]}$	p=0.25				
		$\rightarrow$ I-Track2$_{[f]}$	p=0.25				
		$\rightarrow$ P-Track1$_{[f]}$	p=0.10				
		$\rightarrow$ P-Track2$_{[f]}$	p=0.10				
		$\rightarrow$ G-Track1$_{[f]}$	p=0.15				
		$\rightarrow$ G-Track2$_{[f]}$	p=0.15				
4.	I-Track1$_{[f]}$	$\rightarrow$ I-Track1$_{[f,t+1,x+V+A,y+V+A]}$ $T1_{[f]}$	p=1				
5.	I-Track2$_{[f]}$	$\rightarrow$ I-Track2$_{[f,t+1,x+V+A,y+V+A]}$ $T2_{[f]}$	p=1				
6.	P-Track1$_{[f]}$	$\rightarrow$ P-Track1$_{[f,t+1,x+V+A,y+V+A]}$ P-$T1_{[f]}$	p=1				
7.	P-Track2$_{[f]}$	$\rightarrow$ P-Track2$_{[f,t+1,x+V+A,y+V+A]}$ P-$T2_{[f]}$	p=1				
8.	G-Track1$_{[f]}$	$\rightarrow$ G-Track1$_{[f,t+1,x+V+A,y+V+A]}$ G-$T1_{[f]}$	p=1				
9.	G-Track2$_{[f]}$	$\rightarrow$ G-Track2$_{[f,t+1,x+V+A,y+V+A]}$ G-$T2_{[f]}$	p=1				
10.	P-$T1_{[f]}$	$\rightarrow$ $T1_{[f,t,x+O,y+O]}$ $T2_{[f]}$	p=1	11.	P-$T2_{[f]}$	$\rightarrow$ $T2_{[f,t,x+O,y+O]}$ $T2_{[f]}$	p=1
12.	G-$T1_{[f]}$	$\rightarrow$ $GT1_{[f,t,x+O,y+O]}$ $T1_{[f]}$	p=1	13.	G-$T2_{[f]}$	$\rightarrow$ $GT2_{[f,t,x+O,y+O]}$ $T2_{[f]}$	p=1
14.	$T1_{[f]}$	$\rightarrow$ $V1_{[f]}$ $N_{[f]}$	p=1	15.	$T2_{[f]}$	$\rightarrow$ $V2_{[f]}$ $N_{[f]}$	p=1
16.	$GT1_{[f]}$	$\rightarrow$ $GV1_{[f]}$ $N_{[f]}$	p=1	17.	$GT2_{[f]}$	$\rightarrow$ $GV2_{[f]}$ $N_{[f]}$	p=1
18.	$N_{[f]}$	$\rightarrow$ $n_{[f]}$ $N_{[f]}$	p=0.1	19.	$V1_{[f]}$	$\rightarrow$ $G1_{[f]}$ $G3_{[f]}$ $G7_{[f]}$	p=0.4
		$\rightarrow$ $n_{[f]}$	p=0.25			$\rightarrow$ $G1_{[f]}$ $G3_{[f]}$	p=0.3
		$\rightarrow$ λ	p=0.65			$\rightarrow$ $G1_{[f]}$ $G7_{[f]}$	p=0.25
20.	$V2_{[f]}$	$\rightarrow$ $G3_{[f]}$ $G8_{[f]}$ $G12_{[f]}$	p=0.4			$\rightarrow$ λ	p=0.05
		$\rightarrow$ $G8_{[f]}$ $G12_{[f]}$	p=0.3	21.	$GV1_{[f]}$	$\rightarrow$ G-$G1_{[f]}$ G-$G3_{[f]}$ G-$G7_{[f]}$	p=0.2
		$\rightarrow$ $G3_{[f]}$ $G12_{[f]}$	p=0.25			$\rightarrow$ G-$G1_{[f]}$ G-$G3_{[f]}$	p=0.3
		$\rightarrow$ λ	p=0.05			$\rightarrow$ G-$G1_{[f]}$ G-$G7_{[f]}$	p=0.25
22.	$GV2_{[f]}$	$\rightarrow$ G-$G3_{[f]}$ G-$G8_{[f]}$ G-$G12_{[f]}$	p=0.4			$\rightarrow$ λ	p=0.05
		$\rightarrow$ G-$G8_{[f]}$ G-$G12_{[f]}$	p=0.3	23.	$G1_{[f]}$	$\rightarrow$ $S1_{[f]}$ $S2_{[f]}$	p=0.45
		$\rightarrow$ G-$G3_{[f]}$ G-$G12_{[f]}$	p=0.25			$\rightarrow$ $S1_{[f]}$ $S3_{[f]}$	p=0.1
		$\rightarrow$ λ	p=0.05			$\rightarrow$ $S1_{[f]}$ $S4_{[f]}$	p=0.1
24.	$G3_{[f]}$	$\rightarrow$ $S5_{[f]}$ $S7_{[f]}$	p=0.45			$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$	p=0.1
		$\rightarrow$ $S5_{[f]}$ $S6_{[f]}$	p=0.1			$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$ $S4_{[f]}$	p=0.1
		$\rightarrow$ $S6_{[f]}$ $S7_{[f]}$	p=0.1			$\rightarrow$ $S1_{[f]}$	p=0.05
		$\rightarrow$ $S4_{[f]}$ $S5_{[f]}$	p=0.1			$\rightarrow$ $S2_{[f]}$	p=0.05
		$\rightarrow$ $S7_{[f]}$ $S8_{[f]}$	p=0.1			$\rightarrow$ λ	p=0.05
		$\rightarrow$ $S5_{[f]}$	p=0.05	25.	$G7_{[f]}$	$\rightarrow$ $S11_{[f]}$ $S15_{[f]}$	p=0.55
		$\rightarrow$ $S7_{[f]}$	p=0.05			$\rightarrow$ $S11_{[f]}$ $S16_{[f]}$	p=0.43
		$\rightarrow$ λ	p=0.05			$\rightarrow$ λ	p=0.02
26.	$G8_{[f]}$	$\rightarrow$ $S13_{[f]}$ $S18_{[f]}$	p=0.55	27.	$G12_{[f]}$	$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$ $S17_{[f]}$	p=0.45
		$\rightarrow$ $S13_{[f]}$ $S17_{[f]}$	p=0.1			$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$	p=0.25
		$\rightarrow$ $S14_{[f]}$ $S18_{[f]}$	p=0.1			$\rightarrow$ $S7_{[f]}$ $S14_{[f]}$ $S18_{[f]}$	p=0.25
		$\rightarrow$ $S15_{[f]}$ $S17_{[f]}$	p=0.1			$\rightarrow$ λ	p=0.05
		$\rightarrow$ $S13_{[f]}$	p=0.05	29.	G-$G3_{[f]}$	$\rightarrow$ $S5_{[f]}$ $S7_{[f]}$	p=0.2
		$\rightarrow$ $S18_{[f]}$	p=0.05			$\rightarrow$ $S5_{[f]}$ $S6_{[f]}$	p=0.05
		$\rightarrow$ λ	p=0.05			$\rightarrow$ $S6_{[f]}$ $S7_{[f]}$	p=0.05
28.	G-$G1_{[f]}$	$\rightarrow$ $S1_{[f]}$ $S2_{[f]}$	p=0.2			$\rightarrow$ $S4_{[f]}$ $S5_{[f]}$	p=0.05
		$\rightarrow$ $S1_{[f]}$ $S3_{[f]}$	p=0.05			$\rightarrow$ $S7_{[f]}$ $S8_{[f]}$	p=0.05
		$\rightarrow$ $S1_{[f]}$ $S4_{[f]}$	p=0.05			$\rightarrow$ $S5_{[f]}$	p=0.2
		$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$	p=0.05			$\rightarrow$ $S7_{[f]}$	p=0.15
		$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$ $S4_{[f]}$	p=0.05			$\rightarrow$ λ	p=0.25
		$\rightarrow$ $S1_{[f]}$	p=0.2	31.	G-$G8_{[f]}$	$\rightarrow$ $S13_{[f]}$ $S18_{[f]}$	p=0.15
		$\rightarrow$ $S2_{[f]}$	p=0.2			$\rightarrow$ $S13_{[f]}$ $S17_{[f]}$	p=0.05
		$\rightarrow$ λ	p=0.2			$\rightarrow$ $S14_{[f]}$ $S18_{[f]}$	p=0.05
30.	G-$G7_{[f]}$	$\rightarrow$ $S11_{[f]}$ $S15_{[f]}$	p=0.30			$\rightarrow$ $S15_{[f]}$ $S17_{[f]}$	p=0.05
		$\rightarrow$ $S11_{[f]}$ $S16_{[f]}$	p=0.30			$\rightarrow$ $S13_{[f]}$	p=0.2
		$\rightarrow$ λ	p=0.40			$\rightarrow$ $S18_{[f]}$	p=0.25
32.	G-$G12_{[f]}$	$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$ $S17_{[f]}$	p=0.2			$\rightarrow$ λ	p=0.25
		$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$	p=0.2	35.	$n_{[f,t]}$	$\rightarrow$ $S1_{[f]}$	p=0.05
		$\rightarrow$ $S7_{[f]}$ $S14_{[f]}$ $S18_{[f]}$	p=0.25			$\rightarrow$ $S2_{[f]}$	p=0.05
		$\rightarrow$ λ	p=0.35	...	...	...	...
						$\rightarrow$ $S20_{[f]}$	p=0.05

Figure 12.20. Grammar Rules for a Vehicle Tracking Domain

To analytically determine marker and differentiator relationships for a given SNT, A, and a partial result, b, it is necessary to define several relationships between search state A and search state b. To accomplish this, it is necessary to think of a given IDP$_g$ grammar as a definition of a search space in which the states belong to classes corresponding to terminals and nonterminals of the grammar and final states belong to the class of SNTs. For example, in the vehicle tracking grammar from Fig. 11.1 in Chapter 11.1 and reproduced in Fig. 12.20, the nonterminal V2 defines a class of states in the search space. During problem solving, there could be many instances of the class, each associated with a different vehicle 2 location. Given the correspondence between elements of an IDP$_g$ grammar and states in a search space, it is possible to compute relationships between states by computationally determining relationships between elements of the grammar.

The following definitions are needed to formally define the concepts of marker and differentiator.

Definition 12.4.1 $D(A) = P(S \vdash^* A)$, *where* $A \in \{SNT\}$. $D(A)$ *defines domain specific frequency distribution functions for the set of SNTs, i.e.,* $D(A)$ *= probability of the domain event corresponding to interpretation A occurring. In general, these distributions will be represented with production rules of the grammar associated with the start symbol. The RHSs of these rules will be from the grammar's set of SNTs (i.e.,* $A \in \{SNT\}$*). The variance associated with this distribution leads to problem solving uncertainty.*

For example, the SNTs for the vehicle tracking grammar are I-Track1, I-Track2, G-Track1, G-Track2, P-Track1, and P-Track2. From production rules 2 and 3 (see Fig. 4.27 on page 84), the domain specific distribution function for the SNTs can be computed from the frequency of the nonterminal Track multiplied by the values for ψ corresponding to each of the SNTs. For example, $D(\text{I-Track1}) = 1.11 * 0.25 = 0.275$. Similarly, $D(\text{I-Track2}) = 0.275$, $D(\text{G-Track1}) = 0.11$, $D(\text{G-Track2}) = 0.11$, $D(\text{P-Track1}) = 0.165$, and $D(\text{P-Track2}) = 0.165$.

If the values for ψ in production rule 2 were to change, these values would be different. For example, if the ψ values for the RHSs of rule 2 were both 0.5, the distributions would be $D(\text{I-Track1}) = 2 * 0.25 = 0.5$, $D(\text{I-Track2}) = 0.5$, $D(\text{G-Track1}) = 0.2$, $D(\text{G-Track2}) = 0.2$, $D(\text{P-Track1}) = 0.3$, and $D(\text{P-Track2}) = 0.3$.

Note that the frequency of an individual SNT can be greater than 1 due to domain ambiguity and to multi-track scenarios.

Definition 12.4.2 $\{RHS(A)\}$ *= the set of elements that appear on right-hand-sides of production rules with A on the left-hand-side,* $A \in N \cup SNT$.

From production rule 20, $\{RHS(V2)\} = \{G3, G8, G12\}$. A more complex rule is 26, $\{RHS(G8)\} = \{S13, S14, S15, S17, S18\}$.

Definition 12.4.3 $P(b \in \{RHS(A)\}^{\vdash^+}) = \sum_{\forall i} P(b \in RHS_i(A)) + \sum_{\forall r'} (P(r' \in \{RHS(A)\}) * P(b \in \{RHS(r')\}^{\vdash^+}))$, *where* $\{RHS(A)\}$ *is the set of all RHSs of A,* $RHS_i(A)$ *is the* i^{th} *potential RHS of production rule of A,* $P(b \in RHS_i(A)) = \psi(RHS_i(A))$ *if* $b \in RHS_i(A)$*, 0 otherwise, each element* r' *is a nonterminal that appears in a RHS of A that does not also include b,* $b \in V \cup N \cup SNT$*, and* $A \in N \cup SNT$*. The probability of partial interpretation b being included in any RHS of A, as defined by the distribution function* $\psi(A)$*. The "*$\vdash^+$*" notation indicates that the definition of RHS is recursive. i.e.,* $\{RHS(A)\}^{\vdash^+}$ *represents the transitive closure of all states that can be generated from A. Thus, b can be in an RHS of A, or in the RHS of some element of an RHS of A, etc.*

From production rules 20, 24, 26, and 27,

$$\{RHS(V2)^{\vdash^+}\} = \{G3, G8, G12, S4, S5, S6, S7, S8, S13, S14, S15, S17, S18\} \quad (12.1)$$

$$P(S4 \in \{RHS(V2)\}^{\vdash^+}) = \quad (12.2)$$

$$P(G3 \in \{RHS(V2)\}) * P(S4 \in \{RHS(G3)\}^{\vdash^+}) = 0.65 * 0.1 = 0.065 \quad (12.3)$$

Definition 12.4.4 $P(A \vdash^* b) = D(A) * P(b \in \{RHS(A)\}^*), A \in SNT, b \in V \cup N$. *Probability that the partial interpretation, b, is generated from full or partial interpretation A, where b is a descendant of A.*

Definition 12.4.5 *Ambiguity – Given a domain event, A, its interpretation is* ambiguous with *the interpretation of a second domain event, B, when B* subsumes *A (the subsume relationship is specified in Chapter 4). i.e., A is ambiguous with B when the low-level signal data generated by B can be mistaken for an A. Note that this definition of ambiguity is* not *reflexive. Thus, A being ambiguous with B does not imply that B is ambiguous with A.*

Definition 12.4.6 $P(A \cap b) = P(A \vdash^* b) + \sum_{\forall B} P(B \vdash^* b), A, B \in SNT, b \in V \cup N$, *and where the interpretation of A is* ambiguous with *the interpretation of each B. Intersection of domain events A and b, where b is a descendant of A. The intersection of A and b will occur when both A and b are generated during the course of a specific problem solving instance. This will occur when A leads to the generation of b and when the occurrence of a distinct event, B, leads to the generation of b and when A is ambiguous with B. In the case where B leads to the generation of b, b and A still intersect because an A will be generated during processing since A is ambiguous with B.*

Definition 12.4.7 $P(b) = \sum_{\forall A} D(A) * P(A \cap b \mid A), A \in SNT, b \in V \cup N \cup SNT$. *The probability of partial interpretation b being generated. The notation $P(b \cap A \mid A)$ is the probability that b and A intersect in situations where event A is responsible for the generation of the signal data.*

Given these definitions, it is now possible to formally define the concept of marker:

Definition 12.4.8 $MARKER(A, b) = \frac{P(A \cap b)}{D(A)}$, *where A is an element of the set SNT and b is any terminal or nonterminal. The value of the MARKER function indicates the probability that the generation of an A will cause the generation of a b. A value of MARKER(A, b) that is close to 1 indicates a relationship where every occurrence of A indicates that a b can be derived. The value of MARKER(A, b) indicates the strength of the relationship between A and b.*

Knowledge of markers can be used to design a problem solving architecture and to construct focusing mechanisms in a dynamic control strategy. Intuitively, if you have a strong marker for an SNT, and the marker is not present, then you can eliminate the SNT from further consideration with known risk. For example, assume that the vehicle tracking system is attempting to extend a track of type I-Track1 through a large amount of noise spread over a wide region. This could be computationally expensive if the problem solver has to examine the implications of every possible point of noise in a bottom-up fashion. The number of potential tracks, and the cost, will increase combinatorially with the amount of noise. Information about markers can be used to design control strategies to reduce search costs in these situations. In

the sample grammar from Fig. 12.20, the strongest marker at the group level for an individual component of an I-Track1 (i.e., a vehicle location, V1) is G1. Specifically,

$$MARKER(V1, G1) = P(V1 \cap G1)/P(V1) = \tag{12.4}$$

$$(P(V1) * P(G1 \in \{RHS(V1)\}^{\vdash^+}))/P(V1) = P(G1 \in \{RHS(V1)\}^{\vdash^+}) = 0.95. \tag{12.5}$$

This information could be used to design an expectation driven control strategy for extending an I-Track1 (from t_i to t_j) by predicting the characteristics of all G1's that can be used to extend the track to time t_{j+1} and then only following the implications of the G1's that match the predictions. Also, this information could be used in differential diagnosis processing to disambiguate competing hypotheses [Carver and Lesser, 1993]. For example, in certain situations it is possible to use marker information to determine if a hypothesis was erroneously derived from noise. By examining a hypothesis' supporting data, a problem solver can determine if the support from a strong marker is consistent with expectations. If the support is either much less or much greater than expected, this would be a good indication that the hypothesis is not correct.

For example, assume that, for hypothesis X, $MARKER(X, y) = 0.99$. If the problem solver generates an X for which there is no y supporting data, the problem solver can conclude that the probability of this happening is $1 - 0.99$ and that the X might be erroneous. In the vehicle tracking domain, this might occur if there is a signal group, s, that is a strong marker for a particular kind of track, t. If the problem solver can derive an interpretation of a t without finding any s signals, it is likely that the hypothesis is incorrect.

In contrast, the signal data S15 is a poor marker for the group level event G8.

$$MARKER(G8, S15) = P(G8 \cap S15)/P(G8) = \tag{12.6}$$

$$(P(G8) * P(S15 \in \{RHS(G8)\}^{\vdash^+}))/P(G8) = P(S15 \in \{RHS(G8)\}^{\vdash^+}) = 0.1. \tag{12.7}$$

A formal definition of the differentiator relationship can be given as:

Definition 12.4.9 *DIFF(*A, b*)* $= \frac{P(A \cap b)}{P(b)}$ *where* $A \in SNT$*, and* $b \in V \cup N \cup SNT$ *is any terminal or nonterminal. A value of DIFF(*A, b*) that is close to 1 indicates a strong causal relationship between* A *and* b *to the exclusion of all other causes. A value of DIFF(*A, b*) that is close to 0 indicates a weak causal relationship between* A *and* b*.*

Knowledge about differentiators can be used both in the design of problem solving architectures and dynamic control algorithms. Architecturally, differentiators can be used to construct special operators for differential diagnosis [Carver and Lesser, 1993]. In control algorithms, differentiators can be used to focus problem solving activity [Erman *et al.*, 1980].

In the vehicle tracking grammar, S1, S2, and S3 are strong differentiators for the event V1 (a vehicle location of type 1). Specifically,

$$DIFF(V1, S1) = P(V1 \cap S1)/P(S1) = \tag{12.8}$$

$$(P(V1) * P(S1 \in \{RHS(V1)\}^{\vdash^+}))/P(S1) = 0.55 * 0.70/0.39 = 0.99. \tag{12.9}$$

(Note that S1 is also generated by production rule 35, and, consequently, $P(S1)$is greater than the probability of S1 being derived solely from V1.) Similarly, DIFF($V1, S2$) $= 0.55 *$

$0.70/0.39 = 0.99$ and DIFF$(V1, S3) = 0.55 * 0.3/0.17 = 0.97$. Given a large amount of noise, a possible control strategy would be to determine if there was a large amount of S1, S2, and S3 events in the data. If there are, there is a strong likelihood that they were generated by a track with V1 as a component. This information could be used to filter out data that are weak markers for V1. For example, MARKER(V1,S8) = 0.07. Consequently, ignoring S8 data might be a reasonable strategy in this situation.

In general, the marker and differentiator relationships can be used to explain the success of techniques such as approximate processing and incremental planning [Carver and Lesser, 1991]. From a bottom-up perspective, the differentiator relationship can identify the intermediate results that are most appropriate for abstracting and clustering. From a top-down perspective, the marker relationship can help predict the expected characteristics of intermediate results derived from model-driven processing and thus focus processing by filtering out noise.

Though not discussed in this thesis, the concepts of marker and differentiator can be further refined to include information about the credibilities of the hypotheses. For example, the relationships could vary significantly for low-credibility events and high-credibility events. Given a track level hypothesis that has a low-credibility, its relationship with markers could be quite different from that of a hypothesis with similar characteristics and a high-credibility. The same is true for differentiator relationships. A low-level hypothesis with a high-credibility may be a much better differentiator than a similar hypothesis with a low-credibility.

Also, a similar strategy could be based on the value of a specific characteristic variable (CV). For example, given the grammar rule n.$A \rightarrow C$, the grammar could be modified by adding the rules, n.1.$A \rightarrow C_s$ and n.2.$A \rightarrow C_{\not s}$, where s is a specific value for a CV and each rule has a corresponding distribution value $\psi_{n.1}$ and $\psi_{n.2}$, respectively. In the vehicle tracking domain, s could be a particular location in a sensed region, an energy level, a vehicle type, a velocity, etc. Given these new rules, MARKER(A, C_s) would then represent the degree to which the CV value s is a marker for A.

12.5 Architectural Design Issues

In the design of a problem solving architecture, markers can play a significant role in the specification of the operators of Ω, ω, and Φ. Constructive search spaces are often associated with opportunistic problem solving strategies that can use any of the low-level data as a starting point for the derivation of an interpretation. This is also referred to as *island driving* [Erman *et al.*, 1980]. As will be discussed in Chapter 12.7, this technique can result in significant gains in efficiency. However, opportunistic search can also result in large amounts of redundancy [Lesser *et al.*, 1989b].

Markers can be used to reduce this uncertainty by identifying paths that can be pruned a priori without the risk of eliminating the ability to connect the search path. This can be a very effective strategy in domains that have certain kinds of redundant search paths. However, for various reasons, it *may not be advantageous to do this.* For example, in some domains, this may restrict the ability of a problem solver to do timely pruning. This can happen in situations where a problem solver is using dynamic pruning and where the eliminated paths would have lead to a final solution more quickly and where the final solution could have been used to prune paths that were not pruned by other means.

The use of markers in the design of an architecture will be illustrated using the IDP grammar shown in Fig. 12.21 on page 240. The IDP$_G$ version of this grammar is shown in Fig. 12.22. This grammar was originally described in Chapter 3. The SNTs of this grammar

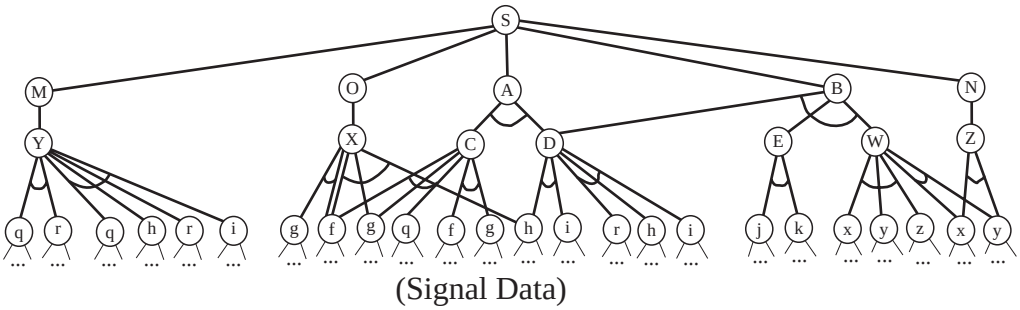


Figure 12.21. Interpretation Grammar G

are A, B, M, N, and O. For this example, the initial base space operators are those shown in Fig. 12.23. These are derived directly from the rules of the generative grammar. The rules corresponding to $b \rightarrow (signaldata)$, rules 7 - 15 of Fig. 12.22, are not included in the set of base space operators as they will not be used in this example.

Observation of the generative grammar indicates that there are a number of relationships in this grammar for which the MARKER function returns a value of 1. For example, both MARKER(M, q) and MARKER(M, r) have values of 1. Similarly, MARKER(O, f), MARKER(O, g), MARKER(A, f), and MARKER(A, g) all have values of 1. Note the relationships between SNTs A and O and the intermediate results f and g. The f and g intermediate results are markers for both SNTs.

The basic control cycle shown in Fig. 1.2 indicates that an operator is instantiated for *each* of the partial results to which it can be applied. Thus, given intermediate results q, h, r, and i, four instantiations of operator 17 would be created, one for each of the four intermediate results. Obviously, this level of redundancy is excessive and leads to large and unnecessary increases in problem solving cost. (This sentence is redundant but necessary.) One solution is to add precondition operators or the goal processing operators from Chapter 4. Another alternative is to modify the operators so that they are only applied to the intermediate states with marker values of 1. This is represented as shown in Fig. 12.24. The parentheses indicate intermediate results to which the operator is not applied.

There is a significant problem with this approach. If the data in parentheses is not available when the operator is applied, the operator will fail in situations where it should succeed. Or it will generate a result with lower credibility, missing or incorrect characteristics, etc. This is not a problem if the data in parentheses is the "low-level" data that is part of the start state, $\mathcal{S}$. This is because low-level signal data cannot be generated at a later point in problem solving (unless the problem solver/problem domain receives signal data incrementally). Otherwise, additional measures are necessary to insure that it is still possible to connect the search space after deleting paths.

In the example in Fig. 12.25, there are multiple derivation paths resulting from the interaction between the recursive rule $A \rightarrow AA$ and the other rules. These interactions result in redundancy. The base interpretation grammar is shown in Fig. 12.26.

Assuming that all interpretations of a given input are identical regardless of the rules used to generate them, significant efficiencies can be gained by eliminating some of the rules from

Interpretation Grammar *G'*

grammar rule	distribution	credibility	cost
0.1 $S \rightarrow A$	$\psi(0.1) = 0.2$	$f_{0.1}(f_A)$	$g_{0.1}(g_A)$
0.2 $S \rightarrow B$	$\psi(0.2) = 0.2$	$f_{0.2}(f_B)$	$g_{0.2}(g_B)$
0.3 $S \rightarrow M$	$\psi(0.3) = 0.2$	$f_{0.3}(f_M)$	$g_{0.3}(g_M)$
0.4 $S \rightarrow N$	$\psi(0.4) = 0.2$	$f_{0.4}(f_N)$	$g_{0.4}(g_N)$
0.5 $S \rightarrow O$	$\psi(0.5) = 0.2$	$f_{0.5}(f_O)$	$g_{0.5}(g_O)$
1. $A \rightarrow CD$	$\psi(1) = 1$	$f_1(f_C,f_D, \Gamma_1(C,D))$	$g_1(g_C,g_D,C(\Gamma_1(C,D)))$
2. $B \rightarrow DEW$	$\psi(2) = 1$	$f_2(f_D,f_E,f_W, \Gamma_2(D,E,W))$	$g_2(g_D,g_E,g_W,C(\Gamma_2(D,E,W)))$
3.0 $C \rightarrow fg$	$\psi(3.0) = 0.5$	$f_{3.0}(f_f, f_g, \Gamma_{3.0}(f,g))$	$g_{3.0}(g_f, g_g,C(\Gamma_{3.0}(f,g)))$
3.1. $C \rightarrow fgq$	$\psi(3.1) = 0.5$	$f_{3.1}(f_f,f_g,f_q, \Gamma_{3.1}(f,g,q))$	$g_{3.1}(g_f,g_g,g_q,C(\Gamma_{3.1}(f,g,q)))$
4. $E \rightarrow jk$	$\psi(4) = 1$	$f_4(f_j,f_k, \Gamma_4(j,k))$	$g_4(g_j,g_k,C(\Gamma_4(j,k)))$
5.0 $D \rightarrow hi$	$\psi(5.0) = 0.5$	$f_{5.0}(f_h,f_i, \Gamma_{5.0}(h,i))$	$g_{5.0}(g_h,g_i,C(\Gamma_{5.0}(h,i)))$
5.1. $D \rightarrow rhi$	$\psi(5.1) = 0.5$	$f_{5.1}(f_r,f_h,f_i, \Gamma_{5.1}(r,h,i))$	$g_{5.1}(g_r,g_h,g_i,C(\Gamma_{5.1}(r,h,i)))$
6.0 $W \rightarrow xyz$	$\psi(6.0) = 0.5$	$f_{6.0}(f_x,f_y,f_z, \Gamma_{6.0}(x,y,z))$	$g_{6.0}(g_x,g_y,g_z,C(\Gamma_{6.0}(x,y,z)))$
6.1. $W \rightarrow xy$	$\psi(6.1) = 0.5$	$f_{6.1}(f_x,f_y, \Gamma_{6.1}(x,y))$	$g_{6.1}(g_x,g_y,C(\Gamma_{6.1}(x,y)))$
7. $f \rightarrow (s)$	$\psi(7) = 1$	$f_7(f_{(s)}, \Gamma_7((s)))$	$g_7(g_{(s)},C(\Gamma_7((s))))$
8. $j \rightarrow (s)$	$\psi(8) = 1$	$f_8(f_{(s)}, \Gamma_8((s)))$	$g_8(g_{(s)},C(\Gamma_8((s))))$
9. $g \rightarrow (s)$	$\psi(9) = 1$	$f_9(f_{(s)}, \Gamma_9((s)))$	$g_9(g_{(s)},C(\Gamma_9((s))))$
10. $k \rightarrow (s)$	$\psi(10) = 1$	$f_{10}(f_{(s)}, \Gamma_{10}((s)))$	$g_{10}(g_{(s)},C(\Gamma_{10}((s))))$
11. $h \rightarrow (s)$	$\psi(11) = 1$	$f_{11}(f_{(s)}, \Gamma_{11}((s)))$	$g_{11}(g_{(s)},C(\Gamma_{11}((s))))$
12. $x \rightarrow (s)$	$\psi(12) = 1$	$f_{12}(f_{(s)}, \Gamma_{12}((s)))$	$g_{12}(g_{(s)},C(\Gamma_{12}((s))))$
13. $i \rightarrow (s)$	$\psi(13) = 1$	$f_{13}(f_{(s)}, \Gamma_{13}((s)))$	$g_{13}(g_{(s)},C(\Gamma_{13}((s))))$
14. $y \rightarrow (s)$	$\psi(14) = 1$	$f_{14}(f_{(s)}, \Gamma_{14}((s)))$	$g_{14}(g_{(s)},C(\Gamma_{14}((s))))$
15. $z \rightarrow (s)$	$\psi(15) = 1$	$f_{15}(f_{(s)}, \Gamma_{15}((s)))$	$g_{15}(g_{(s)},C(\Gamma_{15}((s))))$
16. $M \rightarrow Y$	$\psi(16) = 1$	$f_{16}(f_Y)$	$g_{16}(g_Y)$
17.0 $Y \rightarrow qr$	$\psi(17.0) = 0.5$	$f_{17.0}(f_q,f_r, \Gamma_{17.0}(q,r))$	$g_{17.0}(g_q,g_r,C(\Gamma_{17.0}(q,r)))$
17.1 $Y \rightarrow qhri$	$\psi(17.1) = 0.5$	$f_{17.1}(f_q,f_h,f_r,f_i, \Gamma_{17.1}(q,h,r,i))$	$g_{17.1}(g_q,g_h,g_r,g_i,C(\Gamma_{17.1}(q,h,r,i)))$
18. $N \rightarrow Z$	$\psi(18) = 1$	$f_{18}(f_Z)$	$g_{18}(g_Z)$
19. $Z \rightarrow xy$	$\psi(19) = 1$	$f_{19}(f_x,f_y, \Gamma_{19}(x,y))$	$g_{19}(g_x,g_y,C(\Gamma_{19}(x,y)))$
20. $O \rightarrow X$	$\psi(20) = 1$	$f_{20}(f_X)$	$g_{20}(g_X)$
21.0. $X \rightarrow fgh$	$\psi(21.0) = 0.5$	$f_{21.0}(f_f,f_g,f_h, \Gamma_{21.0}(f,g,h))$	$g_{21.0}(g_f,g_g,g_h,C(\Gamma_{21.0}(f,g,h)))$
21.1. $X \rightarrow fg$	$\psi(21.1) = 0.5$	$f_{21.1}(f_f,f_g, \Gamma_{21.1}(f,g))$	$g_{21.1}(g_f,g_g,C(\Gamma_{21.1}(f,g)))$

(s) = signal data $\Gamma_n(i,j,...)$ = semantic evaluation function for rule n $C(\Gamma_n(i,j,...))$ = cost of executing $\Gamma_n(i,j,...)$

Figure 12.22. Interpretation Grammar *G* with Fully Specified Distribution, Credibility, and Cost Functions

1. S → A
2. S → B
3. S → C
4. S → D
5. S → E
6. A → CD
7. B → DEW
8. C → fg
9. C → fgq
10. E → jk
11. D → hi
12. D → rhi
13. W → xyz
14. W → xy
15. M → Y
16. Y → qr
17. Y → qhri
18. N → Z
19. Z → xy
20. O → X
21. X → fgh
22. X → fg

Figure 12.23. IDP$_i$, Base Space Operators for Grammar G

8. C → f(g)
9. C → f(gq)
16. Y → q(r)
17. Y → q(hri)
21. X → f(gh)
22. X → f(g)

Figure 12.24. Modified Base Space Operators for Grammar G

1. S → A A
2. S → A T
3. A → A T
4. A → A A
5. A → T T
6. T → u | v | w | x | y | z

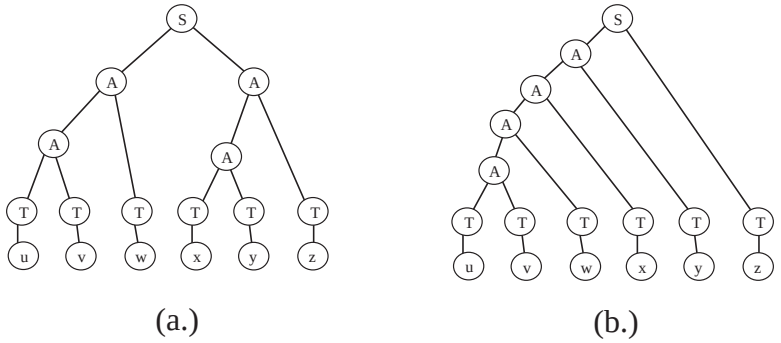


Figure 12.25. Grammar R and Redundant Interpretations for Input "uvwxyz"

1.	S	→ A A
2.	S	→ A T
3.	A	→ A T
4.	A	→ A A
5.	A	→ T T
6.	T	→ u
7.	T	→ v
8.	T	→ w
9.	T	→ x
11.	T	→ y
12.	T	→ z

Figure 12.26. Base Space Operators for Grammar R

the set of base space operators. The issue is then to decide which operators to remove that will still allow the search space to be connected and that will result in an increase in problem solver efficiency. Markers can be used to guide this process. One simple heuristic is the following: If the removal of a rule does not restrict a problem solver's ability to connect both signal data to a strong marker and the marker to the set of SNTs, then the rule is a good candidate for removal.

In this example, the nonterminal A is a strong marker for the single SNT, S since MARKER$(S, A) = 1$. As a consequence, rule 4 is a good candidate for elimination. Eliminating this rule will not preclude the problem solver from connecting any of the signal data to A. Nor will it preclude the generation of S from A.

This example is simple enough that it is possible to identify strong markers by inspection. It is also possible to identify candidate rules for elimination and to check if the search space is still connected after their removal. In more complex domains, it is necessary to construct a connectivity matrix and take the transitive closure of it to insure that the search space is still connected.

Use of the ***Arity Texture*** demonstrated for a vehicle tracking domain. A potential problem solution consists of one vehicle location from each of the goals.

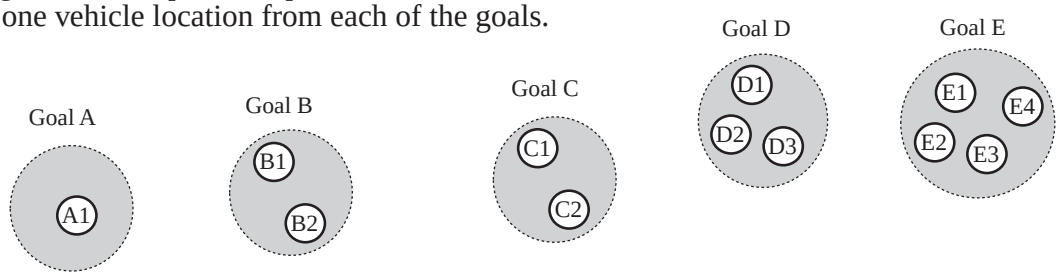


Figure 12.27. Arity Example

12.6 Dynamic Control Design Issues - Estimating *UPC* Values

In some domains, it is feasible to use the a priori calculation of MARKER(A, b) as an estimate $\hat{P}$ instead of the dynamic calculation of the precise value of the probability vector, P, in the *UPC* vectors, where P is the probability of reaching a particular SNT from a state. If there is a relatively small amount of ambiguity, this approximation will be fairly accurate. However, if there is a significant amount of ambiguity or if the problem solver uses pruning operators, the estimate could be too erroneous to be of any use.

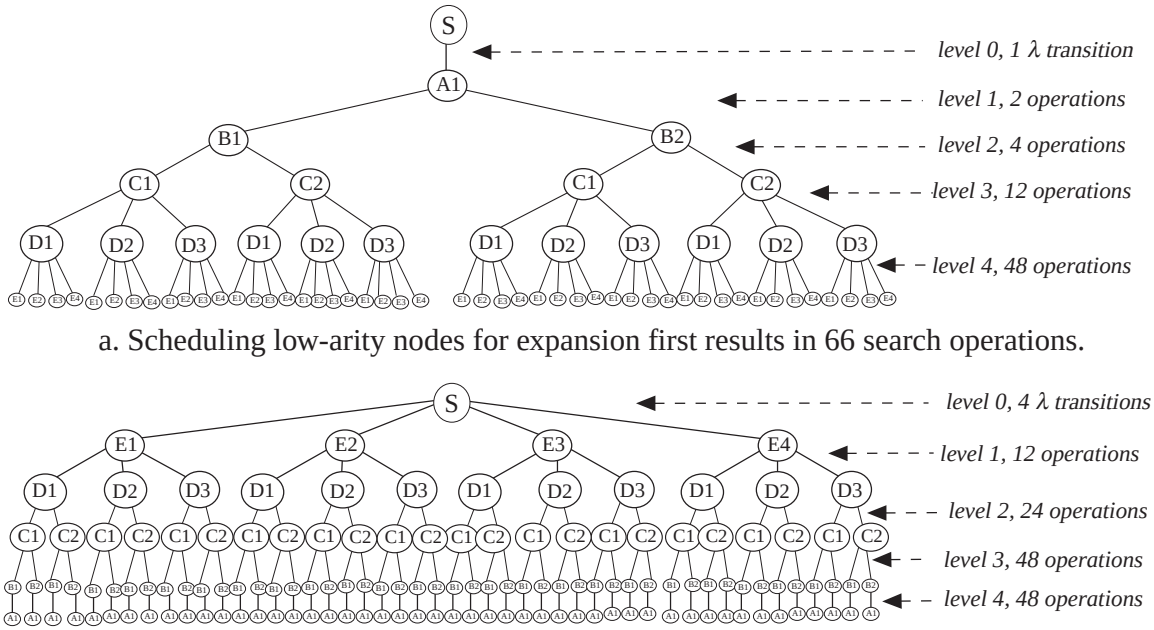
The intersection relationship was defined in Definition 12.4.6 and in Chapter 12.4. Intersection is similar to the concept of markers but it is stronger in the sense that it is bidirectional. For example, given $P(A \cap b) = 1$, then every occurrence of an A will result in the occurrence of a b. Furthermore, the derivation of a b will indicate that it is possible to derive an A. However, like markers, an intersection value of 1 does not rule out the possibility that a third event, C, caused both the A and the b.

The expected costs associated with paths from intermediate states to SNTs, $\hat{C}$, can be estimated from a priori expectations derived from the IDP$_g$representation of the domain. In domains that do not use pruning, this can be a relatively accurate estimate. Even in domains that use pruning, the estimate can be adjusted to account for the pruning. This will result in greater variance, but it might still be a reasonable estimate to use in control decisions.

12.7 Arity

The IDP/*UPC* framework can be used to extract a variety of features from a domain. One such feature is what we will call *arity*. Issues related to arity are associated with the frequency of certain domain events and the related implications for the branching factor of the resulting search space. As many researchers including Fox and Kanal have noted [Fox, 1983, Kumar and Kanal, 1988], the order in which a problem solver attempts to solve subproblems can have a significant impact on the overall cost of problem solving. In general, if a search space is viewed as a tree, it is most efficient to solve the subproblems with the smallest branching factors (or arity) first.

For example, consider the situation shown in Fig. 12.27. This figure represents an interpretation problem domain in which the problem solver attempts to track a vehicle moving through multiple time periods. In the example, each time period is represented by a goal which includes multiple possibilities for the vehicle's position in that time period.



a. Scheduling low-arity nodes for expansion first results in 66 search operations.

b. Scheduling high-arity nodes for expansion first results in 132 search operations.

Graphical representation of the effects of the ***Arity*** texture.
(Nodes represent search states, arcs represent search operations.)

Figure 12.28. Search Paths for Arity Example

Figure 12.28 represents the the effects of arity on the cost of search-based problem solving. Figure 12.28.a represents the cost of problem solving when the operator applications are arranged in the most efficient order. Figure 12.28.b represents the cost of problem solving when the operator applications are arranged in the least efficient order. As shown, the difference in cost is a factor of two.

Figure 12.29 shows several IDP$_i$ grammars that could specify the problem solving actions for interpreting this domain data. In each grammar example, "NTx" represents a nonterminal and "F" represents a final state. The lettered elements represent the respective subproblems. As shown, grammars can be used to generate virtually any sequence of problem solving events. It is important to recognize that we assume the subproblems are independent and can therefore be solved in any order. The interpretation trees in Fig. 12.29 indicate the order in which the problem solving occurs. Thus, Fig. 12.29.a indicates that the problem solver first finds a solution to subproblem A, then B, etc. Fig. 12.29.d indicates the problem solver first finds a solution to subproblem A, then E, then D, etc.

The grammar shown in Fig. 12.29.a generates a left-to-right interpretation of the data. This happens to correspond to the least cost organization of the operators. The grammar in Fig. 12.29.b generates a right-to-left interpretation of the data that corresponds to the greatest cost solution. Figures 12.29.c and 12.29.d generate inside-out and outside-in interpretations with costs that are between the maximum and minimum.

The significance of this example is that it illustrates how the structure of a problem can be exploited to reduce problem solving cost and how this structure can be represented and exploited with a grammar. In the IDP/*UPC* framework, a problem solver that uses an evaluation function

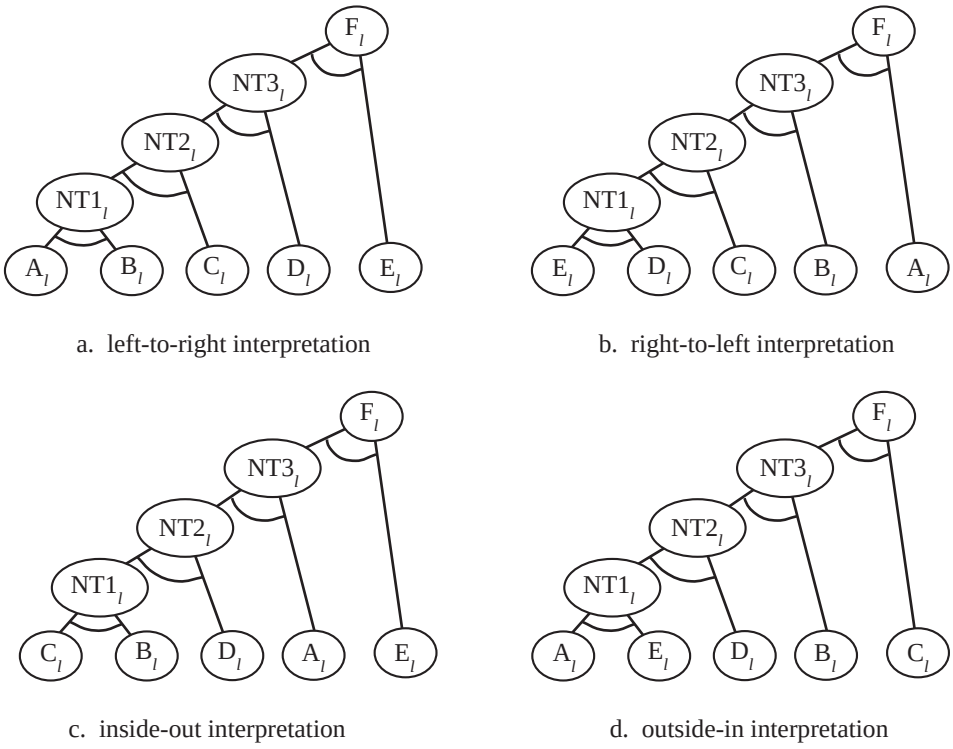


Figure 12.29. Example Grammars for Exploiting Arity Information

based on actual *UPC* values will automatically choose a sequence of operators that minimizes the cost of problem solving. As discussed in this thesis, the cost of implementing a *UPC* based evaluation function is prohibitive and problem solvers that are deployed in real-world situations must use approximations of some kind.

With respect to real-world problems, this example illustrates how the cost of problem solving can be significantly affected by a consideration that might easily be overlooked. For example, in vehicle tracking domains, it is reasonable to expect that the accuracy of sensors will vary over the range of a sensed area. They might be more accurate in the center of the region and less accurate toward the fringes. This often results in the sensor picking up more noise and false signals on the fringe areas and more accurate readings with less noise in the center. Similarly, there may be areas within a sensed region that are known to generate greater than average levels of noise.

In each of these situations, the design of the problem solving grammar can make an important contribution to the overall efficiency of the problem solver. In the case of the sensor's accuracy decreasing toward the fringe of a region, an inside-out grammar can be used to reduce problem arity. In the case of known areas of noise, the grammar can be suitably adjusted to limit the effects of the noise. Designing a grammar so that it exploits the specific statistical properties of a domain, such as arity, to reduce the overall cost of problem is, in effect, embedding a control decision in the grammar.

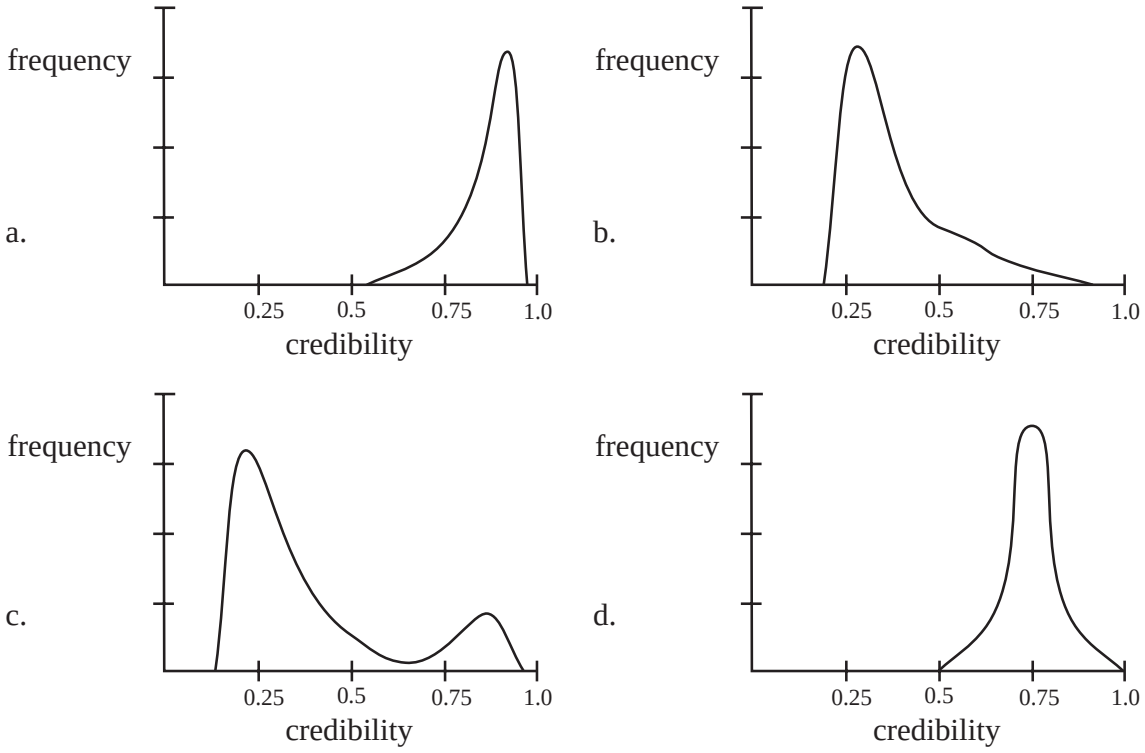


Figure 12.30. Solution Credibility Frequency Map

12.8 Utility Analysis

Given definitions for IDP_g and IDP_i, it is possible to define a map of a grammar's expected utility or credibility values. This is significant because it can be used in the design of sophisticated control mechanisms such as bounding functions for the grammar. As a reminder, bounding functions prune search paths by comparing their credibility based rating to fixed or dynamically determined thresholds. For example, consider the credibility maps shown in Fig. 12.30. These graphs show the expected frequency distribution of the credibility values of interpretations, both correct and incorrect, generated by a problem solver. In each of the graphs, the y axis represents the number of occurrences of a particular frequency. The x axis represents a specific credibility value. In these graphs, and in the domains described in this thesis, an interpretation's credibility spans a range from 0 to 1, where 1 is completely credible. In Fig. 12.30.a, the graph indicates that each problem instance will result in multiple full interpretations. The graph shows the frequency distribution of their frequencies, with most of the interpretations will have high credibilities. If one were designing a problem solver for this domain, this graph would not be encouraging because it indicates that for each problem solving instance, there will be numerous highly-rated possible interpretations. This could indicate that it will be expensive to differentiate these possible interpretations.

In Fig. 12.30.b, most of the interpretations have relatively low credibilities. There are a few that have high frequencies, and it might be possible to use these to prune alternatives while they are still being processed. The fact that the distribution has a greater variance can also be factored into pruning considerations. Distributions with greater variances are more likely

to have outliers that can be used to establish dynamic pruning thresholds, when the outlier is on the higher end of the credibility range, or to establish candidates for pruning, when the outlier is at the lower end. This would be true for the experiments in Chapter 11, but it is not necessarily a valid generalization. Fig. 12.30.c is similar to b but there is a distinctive "hump" of increased frequency toward the higher end of the range. This domain structure could be exploited to construct effective bounding function operators.

Figure 12.30.d shows a distribution that is close to a normal distribution. This is an interesting example to consider because distributions with such well-understood properties can be exploited with predictable effects. For example, given a normal distribution such as this, it is possible to state the probability of a given interpretation being the highest rated solution with a well-defined expected error rate. With this knowledge, a problem solver can make a well-informed decision as to whether or not to continue processing in order to find a higher rated solution or whether to accept a given interpretation as the highest rated, with a known probability of error, and terminate processing.

Credibility maps can also be generated for other elements of the grammar or for the grammar as an aggregate, as shown in Fig. 12.31 and as discussed below. The information shown in the figure can be used for designing bounding functions. Ideally, a bounding function will prune "incorrect" paths (i.e., paths that do not lead to the interpretation with the highest rating) and have no effect on "correct" paths. In a best case scenario, analysis of a domain's utility map can result in the specification of bounding functions that are close to this ideal. Because real-world domains are non-monotonic, as described in Chapter 2, this ideal bounding function is not a realistic goal. However, even in real-world domains the analysis of a utility map can greatly improve the performance of a problem solver and can provide an understanding of a problem solver's behavior, such as a statistically generated expected error rate.

In Fig. 12.31, the x axis is labeled with the names of equivalence classes representing groups of grammar elements. In a vehicle tracking domain, representative equivalence classes would include signal data, group data, vehicle location data, partial tracks of length 2, partial tracks of length 3, and so on. In situations where the elements of a grammar have very individualistic behavior, the equivalence classes can represent specific grammar elements. The y axis represents credibility level. The horizontal line indicates a pruning threshold and the vertical lines above each label on the x axis represents the distribution of credibility values for elements of that class. Elements with values below the pruning threshold correspond to search paths that are eliminated by bounding functions.

Figures 12.31.a and b might represent data from the same domain. Figure 12.31.a might represent the expected credibility distributions of intermediate results that are included on a path to the correct interpretation and 12.31.b might represent intermediate results that are not on a path to the correct interpretation. If this were the case, this would be a very well-behaved domain! As shown, the majority of intermediate results on correct paths have credibilities that are above the threshold while all of the incorrect partial results have credibilities that are below the threshold.

In reality, though, most credibility maps look more like that shown in Fig. 12.31.c. As shown, the credibility distribution of each class lies partly below and partly above the threshold. In most non-monotonic real-world domains, this is true for both correct and incorrect intermediate results. This is why, in large part, problem solving in non-monotonic domains is so difficult.

Figure 12.31.d shows a domain in which only one equivalence class straddles the pruning threshold. This structure could be used in a variety of ways in the design of a problem solver.

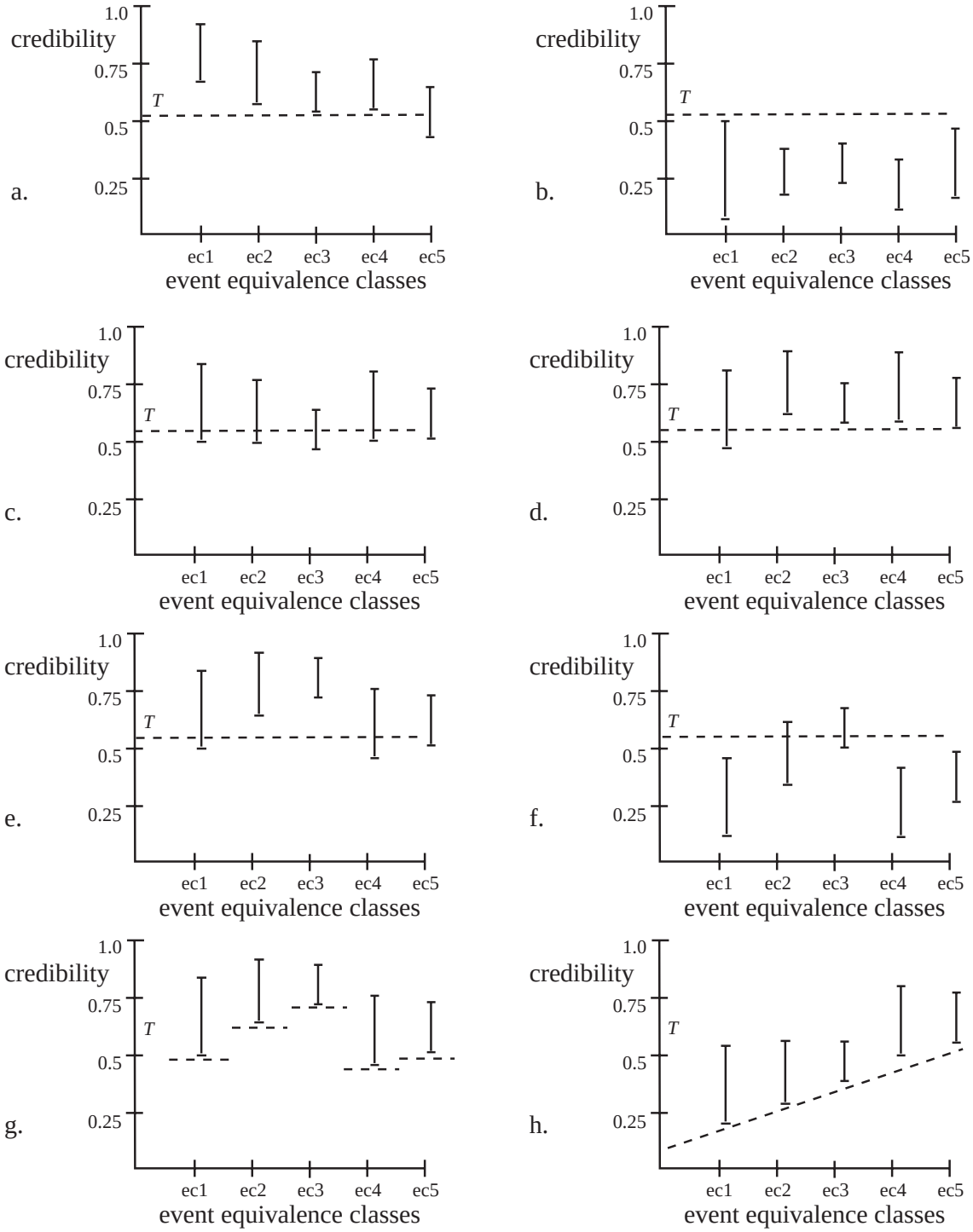


Figure 12.31. Example Frequency Maps

If this map were for correct intermediate results, this might indicate that the problem solver should not apply bounding functions to search paths corresponding to elements from this equivalence class. Alternatively, if this map were for incorrect results, it would indicate that the problem solver should not try to prune equivalence classes 2 - 5 because it will be a waste of effort.

Figures 12.31.e and f illustrate a common problem. Assume that 12.31.e represents the credibility map for correct partial results and f for incorrect results. In the example, a given pruning threshold is effective for some equivalence classes, but not for others. Consequently, it is necessary to design customized pruning threshold for each class. This is shown in Fig. 12.31.g. As shown, it is possible to specify a bounding function that is different for each event equivalence class. For example, the equivalence classes could correspond to partial results stored on different levels of a blackboard. The elements in ec1 might all be on the "signal event" level, etc.

Fig. 12.31.h shows a situation where a very sophisticated bounding function could be used. As shown, assuming that this is a credibility map for correct results, a bounding function could be designed in such a way that the pruning threshold increased or decreased as a function of the equivalence class to which it was applied.

In general, the point of these examples is to illustrate how the IDP specification of a problem domain can be used to design more effective sophisticated control mechanisms.

12.8.1 Calculating Solution Credibility Frequency Maps

Solution credibility frequency maps are generated based on the statistical distribution of the domain credibility factor, μ, the structure of the domain grammar, and the domain specific credibility functions. The calculation is based on the use of the Sample Sets from Definition 5.1.1 in Chapter 5.1.1, and the SNTs for the grammar.

For a given set of SNTs, the solution credibility frequency map for a grammar is defined as:

Definition 12.8.1 $\forall_i \forall_I$ *CMAP(CRED(I)) = CMAP(CRED(I)) +* w_i*, where* i *is an element of* S_{sample}*, I is an interpretation derived for I,* w_i *the weighting of* i *in* S_{sample}*, CRED(I) is the credibility of I, and CMAP is the credibility frequency map. The generation of interpretations from the sample sets assumes a credibility for each sample set equal to the expected credibility of the domain, the domain credibility factor,* μ*.*

Intuitively, this definition takes all the sample sets and all the interpretations that can be constructed for the sample sets and computes the number of times an interpretation of a particular credibility is generated. In the experimental problem solver used in this thesis, the credibility of a solution has a value between 0 and 1. To compute the credibility of a solution of purposes of determining the credibility frequency map, CRED(I) is defined as:

Definition 12.8.2 *CRED(I) = TRUNC((credibility of I) * 100), where TRUNC is the Lisp function "truncate." In essence, the CRED function takes a real value between 0 and 1 and converts it to an integer value between 0 and 100.*

12.8.2 Calculating Domain Credibility Maps

Domain credibility maps are generated based on the statistical distribution of the domain credibility factor, mu, the structure of the domain grammar, and the domain specific credibility functions. The calculation is based on the use of the Sample Sets from Definition 5.1.1 in Chapter 5.1.1.

For a given element, i, of Sample Set, S_{sample}, the problem domain is divided into two sets of partial results, those that are on a paths to the highest rated interpretation for s (These are the correct partial interpretations.), and those that are not on a path to the highest rated interpretation for s (These are the incorrect partial interpretations.).

For a given Sample Set, the domain credibility map for correct partial interpretations is calculated as follows:

Definition 12.8.3 $\forall_i \forall_{ec_j^c}$ *ECRED*$(ec_j^c) = (\forall_{e^c}$ *CRED*$(e^c) * w_i)/ \mid ec_j^c \mid$, *where* i *is an element of* S_{sample}, w_i *the weighting of* $i \in S_{sample}$, ec^c *is the set of event equivalence classes defined for the domain,* ec_j^c *is a specific event equivalence class,* e^c *is an event in* ec_j^c *,* CRED(e^c) *is the credibility of* e^c*, and* ECRED(ec_j^c) *is the map of expected credibilities for the event equivalence class* ec_j^c*. The superscript notation in* ec^c *and* e^c *indicate that these sets only include partial results that are on paths to the full interpretation with the highest credibility. The notation* $\mid ec_j^c \mid$ *represents the cardinality of the set* $\mid ec_j^c \mid$*, i.e., the total number of elements* e^c *in* ec_j^c*. In this computation, an "event" is a specific instantiation of a partial result in a search space. The generation of interpretations from the sample sets assumes a credibility for each sample set equal to the expected credibility of the domain, the domain credibility factor,* μ*.*

For the same Sample Set, the domain credibility map for incorrect partial interpretations is calculated as follows:

Definition 12.8.4 $\forall_i \forall_{ec_j^{c'}}$ *ECRED*$(ec_j^{c'}) = (\forall_{e^{c'}}$ *CRED*$(e^{c'}) * w_i)/ \mid ec_j^{c'} \mid$, *where* i *is an element of* S_{sample}, w_i *the weighting of* $i \in S_{sample}$, $ec^{c'}$ *is the set of event equivalence classes defined for the domain,* $ec_j^{c'}$ *is a specific event equivalence class,* $e^{c'}$ *is an event in* $ec_j^{c'}$ *,* CRED$(e^{c'})$ *is the credibility of* $e^{c'}$*, and* ECRED$(ec_j^{c'})$ *is the map of expected credibilities for the event equivalence class* $ec_j^{c'}$*. The superscript notation in* $ec^{c'}$ *and* $e^{c'}$ *indicate that these sets only include partial results that are* NOT *on paths to the full interpretation with the highest credibility. The notation* $\mid ec_j^{c'} \mid$ *represents the cardinality of the set* $\mid ec_j^{c'} \mid$*, i.e., the total number of elements* $e^{c'}$ *in* $ec_j^{c'}$*. In this computation, an "event" is a specific instantiation of a partial result in a search space. The generation of interpretations from the sample sets assumes a credibility for each sample set equal to the expected credibility of the domain, the domain credibility factor,* μ*.*

Intuitively, these definitions simply compute the average credibilities of the elements of equivalence classes. One definition is for partial results that are on paths to correct interpretations and the other is for partial resuls that are on paths to incorrect partial interpretations. Note that these definitions allow for interactions between the two sets.

12.9 Chapter Summary

This chapter discusses some of the implications and uses of the IDP/*UPC* framework and demonstrates several prototype design tools/theories. It introduces two general analysis

techniques referred to these as *comparative cost analysis* and *constraint flow analysis*. It also discusses some of the implications and uses of the IDP/*UPC* framework and demonstrates several prototype design tools/theories not necessarily related to approximate processing. These include the concept of *marker* and *differentiator* that are based on statistical characteristics of a domain derived from an IDP specification.

CHAPTER 13

EXPERIMENTS WITH APPROXIMATE PROCESSING

We conducted a number of experiments to determine whether or not the analysis framework, in its current state of development, can be applied to domains that use sophisticated control techniques based on approximate processing. Specifically, we wanted to test the framework in a domain where the problem solver's control strategy included all forms of approximate processing discussed in Chapter 10.3. The specific objective was to determine if the analysis tools are capable of representing and analyzing the sophisticated control mechanisms used in problem solvers such as that described by Durfee in [Durfee, 1987]. The experiments that were conducted are described in this section.

The problem solver used for these experiments is based on the grammar introduced in Chapter 12 and shown again in Fig. 13.1. This grammar is somewhat simpler than the grammar used in the experiments in Chapter 11, but it contains many of the same complexities such as noise/missing data and context sensitive attributes linked through the feature list convention.

In the grammar, the elements A, B, and C represent different vehicle types. The feature lists used in these experiments were identical to those used in the experiments in Chapter 11

P.1.1.	T	→ T1 A	p=0.33
P.1.2.	T	→ T2 B	p=0.33
P.1.3.	T	→ T3 C	p=0.33
P.2.	T1	→ A A	p=1.0
P.3.	T2	→ B B	p=1.0
P.4.	T3	→ C C	p=1.0
P.5.	A	→ G1 G2	p=1.0
P.6.	B	→ G3 G4	p=1.0
P.7.	C	→ G5 G6	p=1.0
P.8.1.	G1	→ S1 S2	p=0.9
P.8.2.	G1	→ S1 S2 S6	p=0.1
P.9.	G2	→ S11 S12	p=1.0
P.10.1.	G3	→ S1 S11	p=0.9
P.10.2.	G3	→ S1 S11 S12	p=0.1
P.11.	G4	→ S2 S6	p=1.0
P.12.1.	G5	→ S2 S11	p=0.9
P.12.2.	G5	→ S1 S2 S11	p=0.1
P.13.	G6	→ S6 S12	p=1.0

Figure 13.1. Full Grammar VTG-1 for Tracking Vehicles Through Multiple Time-Locations

AP.1.	$AT_{c1 \cap c2}$	$\rightarrow AT1_{c1}\ VLC_{c2}$
AP.2.	$AT1_{c1 \cap c2}$	$\rightarrow VLC_{c1}\ VLC_{c2}$
C.1.	$VLC_{A \cup B \cup C}$	$\rightarrow A^{lh} \ldots B^{lh} \ldots C^{lh}$
LH.1.	A^{lh}	$\rightarrow$ S1 S2 S6 S11 S12
LH.2.	B^{lh}	$\rightarrow$ S1 S2 S6 S11 S12
LH.3.	C^{lh}	$\rightarrow$ S1 S2 S6 S11 S12
M.G.1.	$\emptyset$	$\rightarrow$ AT

Figure 13.2. Approximations Used to Extend VTG-1

and included time, location (x, y), velocity, acceleration, and energy level (loudness). The semantic functions used were also identical.

The set of base space operators corresponded to the rules of the grammar shown in Fig. 13.1. The meta-level operators used in the problem solver are represented in Fig. 13.2. Three different kinds of operators were used, a level-hopping operator that approximated domain knowledge, a clustering operator that aggregated data in abstract states, and approximate processing operators that generated interpretations in a projection space.

Rules LH.1, LH.2, and LH.3 in Fig. 13.2 are level-hopping operators. Level-hopping takes constraints that would normally span multiple levels of the blackboard in the base space and simplifies and compresses them into a single operator. A more detailed discussion of level-hopping appears in Chapter 10.3.2.

Rule C.1 is a clustering operator that approximates data. This strategy attempts to efficiently apply constraints to aggregations of data and it is useful in domains with large quantities of noise. Instead of processing each individual piece of data, a problem solver aggregates data into an abstract unit and processes the unit as a whole. A discussion of abstract data aggregation appears in Chapter 10.3.1. Rule C.1 combines the results of level hopping into abstract states that are subsequently processed by rules AP.1 and AP.2.

Rules AP.1 and AP.2 are abstract operators in a projection space that mirrors the base space. The results of processing in this space are mapped back to the base space. Ideally, the mapping process reduces the cost of problem solving in the base space by an amount that exceeds the cost of the approximate processing and mapping.

Four experiments were conducted. The first, AE1 (Approximate Experiment 1) did not use any approximate processing. Experiments AE2, AE3, and AE4 used level-hopping, clustering, and approximate processing. The cost of the approximate operators and the mapping functions was varied from 10, to 1, to 50 in the experiments. The results of the experiments are summarized in Table 13.1.

The mapping procedure used in these experiments was the transformation strategy described in Chapter 10.3.4. Given a set of input data, the problem solver first uses level-hopping to generate abstract, vehicle level results. These results are then clustered into an abstract states that are processed as using operators derived from base space operators. The results of this processing are mapped back to the base space by using the results of approximate processing to transform the base space grammar.

The transformation of the base space grammar involves eliminating rules based on the characteristics of the results of approximate processing. For example, the results of approximate

Table 13.1. Summary of Approximate Processing Experiments

Exp	Approx	Approx Cost	$E(C)$	Avg. C	Sig
AE1	none	na	930	927	N
AE2	LH, C, AP	10	337	338	N
AE3	LH, C, AP	1	250	258	N
AE4	LH, C, AP	50	720	723	N

Abbreviations

Exp: Experiment
Approx: Description of approximate operators used by the problem solver.
LH: Level-hopping was used;
C: Clustering was used;
AP: Approximate processing was used.
Approx Cost: Cost assigned to approximate operators in this experiments. All base space operators have cost 10.
$E(C)$: Expected Cost of problem solving.
Avg. C: actual average cost for 100 samples of 50 random problem instances each.
Sig: Whether or not the difference between expected cost and the
actual average cost was statistically significant, Y:yes; N:no.

processing include a track type characteristic. The grammar transformation that takes place eliminates all grammar (and all the associated operators) rules that do not contribute to paths that lead to results of a different track type.

For example, as shown in Fig. 13.3, approximate processing generates an abstract result with properties characteristic of a vehicle track of type A. In the Fig. 13.3, the states labeled with the superscript "LH" represent the results of level-hopping. The states labeled "VLC" represent clusters of vehicle locations. In this case, the clusters are formed using the results of level-hopping. The cluster states are also labeled with a subscript that indicates the conjunction of the vehicle type characteristics of the component states. The results of approximate processing are labeled "AT1" and "AT" for "approximate T1" and "approximate T." Each of the abstract states that is generated by approximate processing has a subscript indicating the vehicle type characteristics encompassed by the state.

Given that the result of approximate processing has a vehicle type characteristic of A, the problem solver can eliminate all operators that lead to results other a vehicle track of type A. The modified grammar is shown in Fig. 13.4. This grammar is then applied to the input data to generate one or more interpretations with no more modifications to the problem solver. (In this particular case, the problem solver will generate four possible solutions.) As will be discussed in the following sections, this approximation/grammar transformation process significantly reduces the cost of problem solver, even in this simple domain.

The following section discuss the changes that were made to the system and the analysis framework to support the analysis of approximate processing and the details of the different sets of experimental results.

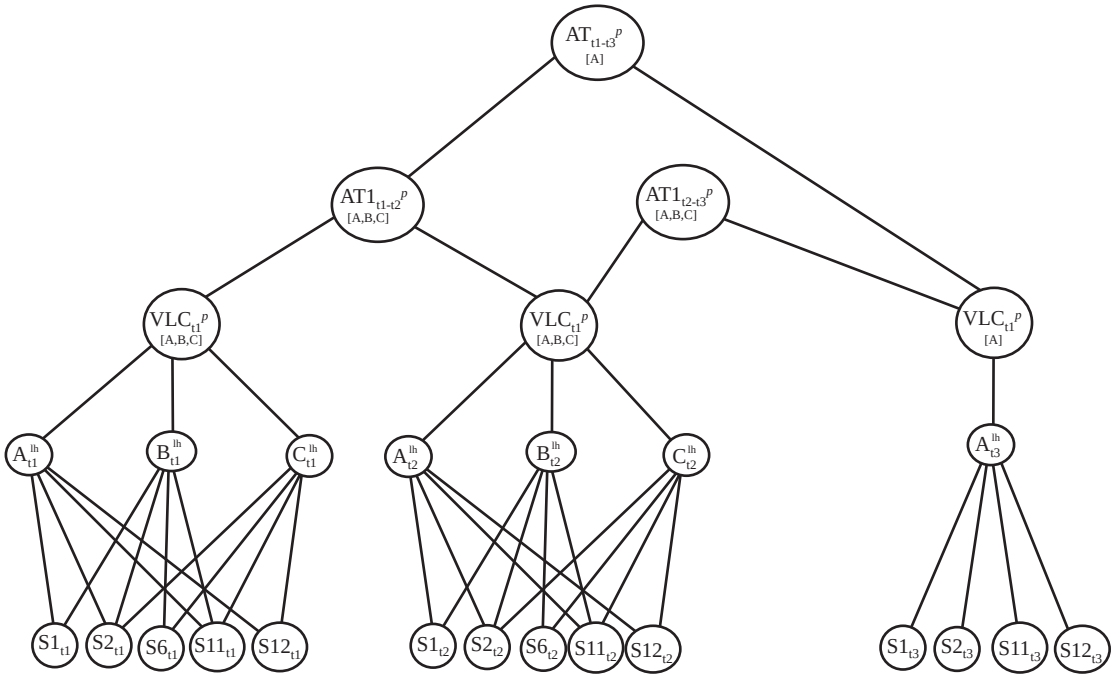


Figure 13.3. Approximate Processing Example

P.1.1.	T	→ T1 A	p=1.0
P.2.	T1	→ A A	p=1.0
P.5.	A	→ G1 G2	p=1.0
P.8.1.	G1	→ S1 S2	p=0.9
P.8.2.	G1	→ S1 S2 S6	p=0.1
P.9.	G2	→ S11 S12	p=1.0

Figure 13.4. VTG-1 Transformed By Mapping Operator

13.1 Modifications

Several modifications were made to the analysis tools and the experimental testbed to support these experiments. The changes to the analysis tools involved incorporating analytical methods for approximate processing that would properly compute the expected frequency of approximate results and the expected costs associated with the approximate results.

Specifically, a base level operator can generate multiple results from a given set of input data. The input to the operator corresponds to the RHS side of a grammar rule and the number of results generated by the operator is a function of the combinatorics of combining the inputs. Thus, if an operator with two inputs has 2 instances of the first input and 3 of the second input, the operator will produce $2 * 3 = 6$ distinct results. This process is described in greater detail in Chapter 5.1 and Appendix D. In contract, clustering operators used in these experiments generate a single result, regardless of the number of inputs.

To account for these differences, the analysis framework was modified by associating a "combining" function to each rule of the grammar. During the computation of frequency maps, the analysis tools use the combining function for each specific grammar rule. This is illustrated in the detailed example in Appendix D.

The computation of cost associated with the approximate processing operators is identical to that for base space operators. However, the cost of problem solving in the base space has been modified to reflect the impact of using transformed grammars. This modification is relatively simple. Instead of computing the cost of problem solving based on the original grammar, the costs are computed based on the transformed grammars. Then the distribution of the transformed grammars is computed from the original base space distributions and this is used to determine a weighting factor that is used to combine the results from the distinct transformed grammars.

The modifications to the experimental testbed are extensive and are described in detail, along with the conceptual issues involved, in Appendix C.

13.2 Experimental Details

The purpose of these experiments was to determine whether or not the analysis tools could accurately predict and explain the performance of a sophisticated control strategy that is based on the use of approximate processing. In addition, these experiments test the viability and validity of the representations we introduced for approximate processing in Chapter 10.3. As shown in Table 13.1, the analysis tools accurately predicted the expected cost of each experiment. In addition, the formulations of approximate processing worked well and were able to reduce the cost of problem solving in a complex problem domain.

Experiment AE1 is the control. AE1 uses no approximate processing, but the expected results were calculated using the modified methods described above in Chapter 13.1. The cost of all problem solving operations in this grammar was 10. The problem solver used a four level blackboard including levels for signal, group, vehicle, partial track, and track data.

In experiments AE2 - AE4, level-hopping, clustering, and approximate processing operators were added, along with a mapping operator that uses the results of approximate processing to transform the base space grammar. The control strategy processed the level-hopping, clustering, approximate processing and mapping before any domain processing was carried out. This was implemented using an evaluation function that rated the meta-level operators higher than any other operator. The mapping function used was a grammar transformation that eliminated

all operators that only generated paths to states that were not consistent with the results of approximate processing, as described above.

In each of the approximate processing experiments, the analysis tools accurately predicted the expected cost of problem solving. In addition, the formulation of approximate processing used functioned accurately and effectively. In each of the experiments, the cost of problem solving was significantly reduced through the use of approximate processing.

13.3 Chapter Summary

This chapter discusses experiments that were conducted to determine whether or not the analysis framework, in its current state of development, can be applied to domains that use sophisticated control techniques based on approximate processing. In all the experiments, the performance predicted by the quantitative analysis tools was consistent with the actual observed results. This verifies that the analysis tools accurately predict the performance of problem solvers that include sophisticated control mechanisms based on abstractions and approximations.

CHAPTER 14

CONCLUSION

This thesis represents a step toward the development of design theories for sophisticated problem solving systems. It experimentally demonstrates that, for an important class of AI problem solving architectures, the class of sophisticated, blackboard-based problem solvers, answers to critical design issues can be related to specific problem domain and problem solver characteristics that are represented in formal, quantifiable ways. The approach used involves developing formal specifications of the characteristics and inherent properties, or *structure*, of problem solvers and the domains in which they are applied and then using this representation to build a quantitative analysis framework for predicting and explaining the performance of a sophisticated problem solver operating in a complex domain. The particular complex domain studied is a signal interpretation task – vehicle monitoring.

In general terms, the most important conclusion that can be drawn from the work presented in this thesis is that the performance of a problem solving system can be analyzed based on formal descriptions of the structure of the domain in which the problem solver is applied and the structure of the sophisticated control architecture used by the problem solver. This analysis can take the form of both predicting the performance levels of the problem solver using analytically derived closed form equations and explaining the performance levels relative to structures in the formal models of the domain and/or the sophisticated control architecture. In addition, the IDP/*UPC* framework can be used for conducting experiments to determine the structure of a specific domain or the objective strategy being used by a problem solver.

The formal approach developed in this thesis, the emphasis it places on the structure of a problem domain, and its relationship to problem solving performance has given new insight into the nature of search-based problem solving. In particular, problem solving and the sophisticated control techniques that are based on the use of abstractions and approximations can now be viewed from a unified perspective. This unified perspective has important implications for analyzing control architectures such as approximate processing, goal processing, and other meta-level control architectures. Using the IDP/*UPC* framework, or other formal approaches, it is possible to compare and contrast alternative architectures in a range of different domains in order to develop an understanding of how a particular architecture might be generalized. This ability could eventually shift the emphasis of AI research to the exploration, and formalization, of the characteristics of natural problem domains.

Furthermore, in developing the IDP/*UPC* framework, this thesis develops a better understanding of sophisticated problem solving and its relationship to other forms of problem solving. The IDP/*UPC* framework was designed intentionally as an extension to the framework introduced by Kanal and Kumar [Kumar and Kanal, 1988] and used to develop a taxonomy of search-based problem solving techniques. By extending this framework to include the sophisticated control mechanisms used in AI problem solving, it is now possible to view many AI control architectures as generalizations of problem solving techniques described formally in other fields, such as operations research.

The following sections review the major contributions that were described in the introduction. For each of the contributions, a detailed summary is given including how the thesis achieved the contribution, where in the thesis the work related to the contribution is described, and the implications associated with the contribution.

14.1 Defining Sophisticated Control, Interpretation Problems, and Complex Domains

Chapter 2 defines the class of problems referred to in this thesis as *sophisticated control problems* and the class of *complex problem domains* and relates both to previous problem and domain definitions. The resulting taxonomy is shown in Fig. 2.2 on page 30. Sophisticated control and complex domains are formally defined in the context of search problems. Specifically, sophisticated control problems are those where, in order to find the statistically optimal operator to execute, the control mechanism must examine the relationships between all possible sets of partial search paths. Complex domains are search problems in which the cost of applying a single operator is potentially exponential with respect to the input to the operator and where the functions for extending search paths are non-monotonic. These definitions establish a formal specification for an important set of AI research problems, they allow these problems to be characterized more precisely, and they enable the results from research on these AI problems to be compared and contrasted to the results of other research efforts in AI and to work in other fields, such as operations research, that have developed search-based formalisms. This chapter also defines the related concepts of *restricted problem domains*, in which search operators have a constant cost, and *local control*, which does not examine any interrelationships between potential problem solving actions.

The IDP/*UPC* analysis framework introduced in this thesis extends work on the analysis of search-based problem solving by eliminating many of the restrictions used previously, such as monotonicity constraints, and by increasing the expressive power of a formal grammar to include elements that are necessary for conducting analysis, such as the specification of *solution nonterminals*, functions associated with a grammar rule's cost, credibility, and probability, and more.

14.2 Formally Specifying Domain Structures and Problem Solving Architectures

Chapter 3 defines the specific class of problems, the *interpretation decision problem (IDP)*, studied in this thesis. An IDP is a constructive search problem where, given an input string of signal data, the problem solver tries to find the *best* interpretation corresponding to the events that could have caused the signal data. IDP problems are defined in terms of discrete optimization problems. Specifically, the problem solver is trying to determine which interpretation from a set of possible interpretations has the highest credibility. This defines the problem in such a way that it is possible to quantitatively analyze it. This is because it provides a clear criterion for termination – problem solving halts when every possible search path has been connected to a final state or eliminated from consideration – and it explicitly represents all meta-level actions that implicitly enumerate (i.e., prune) portions of the search space. At this point the problem solver conducts a linear search of the set of interpretations that were found to determine which has the highest credibility. Also, since it casts interpretation as a discrete optimization problem, it clearly defines interpretation's relationship to other classes of problems and it makes it possible to understand the general applicability of algorithms and other results that are constructed or determined for interpretation domains.

In IDP models, the different feature structures that are defined by domain theories are combined into a unified representation by expressing them in terms of formal grammars and functions associated with production rules of the grammar. Nonterminals of the grammar represent intermediate problem solving states, terminal symbols represent raw sensor input, and the production rules of the grammar represent potential problem solving actions. The grammar rules of IDP models specify the component structure of a domain and each production, p, has associated cost and utility functions, g_p and f_p, that define the cost and utility structures. In addition, IDP models explicitly represent aspects of inherent uncertainty in a domain with the distribution function, ψ, that defines the probability structure of the domain. (i.e., ψ, along with other mechanisms, define inherent uncertainty in a domain.) For a given production, p, the frequency of the occurrence of p's right-hand-side (RHS) is specified by the distribution function $\psi(p)$. Thus, p can have multiple RHSs, RHS_1 through RHS_n, and the distribution of the RHSs is defined by $\psi(p)$. Each production rule, p, is associated with a semantic function, Γ_p that is a function of the subtree components represented by the elements on the right-hand-side of p. Γ_p measures the "consistency" of the semantics of its input data and returns a value that is included in the credibility function. For example, in a speech understanding domain, Γ_p would rate the consistency of the meaning of a sentence and return a value indicating whether or not the sentence made any sense. An IDP grammar also includes an extension to its representation called a *feature list* to represent the characteristics of real-world phenomena.

The framework involves the use of two distinct grammars that reflect the concept of a domain theory and an approximation of the domain theory used in problem solving. These grammars consist of a *generational grammar, IDP*$_G$, and an *interpretation grammar, IDP*$_I$. IDP$_G$ corresponds to the domain theory in the sense that it can be used to generate problem instances that correspond to the actual events that occur in a domain. IDP$_I$ is a representation of the problem solving actions available to a problem solver, including abstract and approximate operators used by the control mechanism. Thus, there is a clear connection between the structure of a problem domain and the associated problem solving architecture.

Chapter 3 defines the four structures that are studied in this thesis, component (or syntax), cost, utility (credibility), and distribution (probability). These structure are defined and illustrated in terms of the *generalized hypergraph* structure shown in Fig. 3.4 on page 46. Chapter 3 also defines the concept of structural interaction. The IDP definition is that it supports the formal specification of an important class of AI problems in such a way that related problem domains and problem solvers can be represented and analyzed quantitatively. As shown in this thesis, this allows comparative studies to be conducted and it is an important tool for prediction and explanation. As such, the IDP definition is an important step toward establishing design theories for AI problem solvers.

The IDP definition also supports the definition of the *convergent search space* concept. An example is shown in Fig. 3.3 on page 45. The convergent search space concept constitutes a formal specification of a class of search spaces that is an important step in establishing design theories for AI problem solvers. Convergent search spaces have characteristic structures that can be exploited in the design of problem solving systems. Specifically, as discussed in Chapters 3 and 4, convergent search spaces include states that are implicitly created during problem solving. These implicit states can be used as candidates or templates for automatically constructing meta-level operators.

Examples of how the IDP formalism modifies a domain grammar to represent phenomena such as noise and missing data are discussed in Chapters 4.1.2 and 4.1.3. Formal definitions for these phenomena are presented in Definitions 4.1.3 and 4.1.6. For example, the definition

of noise is: A grammar, G, is subject to noise iff:
$\exists$ a rule p $\in$ P, where p: A $\rightarrow$ uBv $\Rightarrow$ p$'$: A $\rightarrow n_1$un_2Bn_3vn_4, for u,v $\in$ (V $\cup$ N)*, A $\in$ (SNT $\cup$ N), B $\in$ (V $\cup$ N), $n_i \in$ (V $\cup$ N)*, and $f_p(f_u, f_B, f_v, \Gamma_p(u, B, v)) > f_{p'}(f_{n_1}, f_u, f_{n_2}, f_B, f_{n_3}, f_v, f_{n_4}, \Gamma_{p'}(f_{n_1}, u, f_{n_2}, B, f_{n_3}, v, f_{n_4}))$ for maximum ratings of the f_{n_i}. The distribution of p and p' is modeled by ψ.

An example of the value of these definitions is demonstrated in Definition 4.1.4, which formally specifies the concept of *correlated noise.* Correlated noise is a characteristic phenomenon of interpretation domains that increases the ambiguity of the domain and, as a consequence, the associated cost of problem solving is usually increased as well. This example demonstrates how the IDP formalism can be used to precisely define concepts and phenomena so that they can be clearly understood and analyzed. The definition of correlated noise enables the definition of *uncorrelated noise* and it allows us to explain why, in some situations, adding noise to a domain increases the cost of problem solving and, in other situations, adding noise to a domain does not increase the cost of problem solving. Without formal definitions and representations, such explanations are difficult.

The IDP formalism is also used to specify the concept of *interacting subproblems* in Chapter 4.4. This supports precise definitions and representations of important concepts such as *overlapping goals*, *competing goals*, etc. Given such precise definitions, it is possible to analyze relationships, and their implications, quantitatively. The examples in this chapter are all presented in terms of formal set theory and are related to elements of an IDP grammar. Thus, it is shown how many concepts related to subproblem interaction can be formally defined and studied in terms of an IDP grammar and traditional set theory.

Using these specifications, it is possible to automatically determine and analyze subproblem interactions using an IDP model. The IDP model supports the definition of two important sets, a *component set* (Definition 4.4.1 on page 67) and a *result set* (Definition 4.4.2 on page 67). The component set of a state, s_n, includes all the states that lie on paths from the signal data to s_n. The result set of a state, s_n, in a search space includes all the states that can be reached from s_n. The elements of both sets can be determined from an IDP grammar. Later chapters show how the component set and result set are used in quantitative analysis (Chapter 5) and in the construction of approximate processing techniques (Chapter 10).

An extended definition of *monotonicity* is presented in Chapter 4.5. This definition is an extension of a similar definition from Kanal and Kumar [Kumar and Kanal, 1988] and it clearly distinguishes AI problem domains from other domains in which search-based problem solvers are used. Specifically, Kanal and Kumar use their definition of monotonicity to differentiate between a variety of search techniques. Their differentiation results in a taxonomy of problem solving techniques, each of which is distinguished by the kind of domain in which it can be applied. Given our extended definition of monotonicity, it is now possible to view certain AI search problems from the same perspective and to understand why traditional OR search methods fail in the complex domains typically thought of as "AI domains."

Chapter 4 also shows how sophisticated control mechanisms, such as bounding functions, can be represented in the IDP framework. The approach is similar to that used for representing domain phenomena – sophisticated control mechanisms are represented with extensions to the base grammar. Specifically, given a state, n, for which a bounding function can be defined, every appearance of n on the RHS of a grammar rule in IDP$_i$ is replaced with the nonterminal n^b and the rule $p : n^b \rightarrow n$ is added to the grammar. The knowledge incorporated in the operator corresponding to p is the bounding function on n.

A demonstration of how the IDP formalism represents complex, real-world domains, in this case a vehicle tracking domain, is given in Chapter 4.7. This section of the paper demonstrates how the *feature list convention* developed in this thesis can be used to model the characteristics of real-world phenomena. Specifically, the feature list convention is used to model complex domain events that seem to interact over time and space. The ability to model such interactions is crucial in establishing the applicability of the IDP formalism. The vehicle tracking grammar is extensive and it is shown in Figures 4.25 (page 80), 4.27 (page 84), 4.29 (page 87), and 4.30 (page 89). The example shows how different types of correlated and uncorrelated noise and missing data are modeled in real-world domains and how a sophisticated control mechanism, goal processing, is represented in an IDP$_i$ grammar. The goal processing mechanism is patterned after goal processing mechanisms introduced in Hearsay II [Erman *et al.*, 1980] and the DVMT [Corkill, 1983]. Figure 4.28 on page 85 shows some of the specific problem instances that are generated with this grammar.

The use of the feature list convention is shown in Chapter 4.6. This chapter shows the process that is used to generate problem instances in the experimental testbed developed in this thesis. It demonstrates how subproblem interaction over time and space can be modeled with a context-free grammar. This representation is critical to the IDP/*UPC* framework because the characteristics of context-free grammars support the calculation of the statistical properties of a domain and the development of design and analysis tools, such as those described in Chapter 12. In short, each the feature list convention is used to propagate characteristics and functions of characteristics from a parent node to its children and from a node to its siblings. This suggests that many other real-world domains can be modeled by exploiting a similar approach.

Finally, Chapter 4.8 demonstrates how the IDP formalism can be used in the analysis of a sophisticated control mechanism, bottom-up goal processing. This chapter shows how a control mechanism is represented in the IDP formalism in such a way that it is integrated with the domain processing and can be quantitatively evaluated in the same manner.

14.3 Defining Quantitative Analysis Tools and Methodologies

Chapter 5 defines quantitative analysis tools based on statistical properties of IDP problem and problem solver specifications. These tools can be used to calculate characteristics such as the expected cost of a problem solving instance, the expected frequency with which partial and full interpretations will be generated, the expected utilities of partial and full interpretations, the relative ordering of problem solving actions, and the expected number of correct answers that are eliminated by pruning operators. These measures are significant from an analytical perspective because they measure important properties of a problem solver's performance and they are significant from a conceptual perspective because they demonstrate that relevant quantitative analysis can be conducted using the IDP formalism.

The most important analysis tool defined is the calculation of the complexity of a domain. This is calculated in terms of the expected cost of problem solving for a specific problem instance, $E(C)$. $E(C)$ is measured in terms of computational cost and it represents the cumulative cost of applying all operators required to generate an interpretation. This is a general measure that has several advantages. It is intuitively easy to understand compared to other measures such as expected ambiguity, which is used in the calculation of $E(C)$. $E(C)$ can be used to compare both the performance of a problem solver across different domains or different problem solvers applied to the same domain with units of measure that are consistent.

Most importantly, $E(C)$ represents what is probably the most significant aspect of a problem solver's performance. The definitions in Chapter 5 are verified in the experimental results presented in Chapters 7, 9 and 13.

The basic approach is a three step calculation. The first step calculates the *expected frequency* with which states are generated corresponding to each of the elements of the grammar. (Note that the set of all state frequencies is referred to as the frequency map of the domain.) This step relies primarily on the structure of the domain as specified in the grammar and the distributions associated with the rules of the grammar. The definition of frequency calculation is given in Chapter 5.1.1. The second stage calculates the expected probability with which paths from the states are pruned, which is called the *pruning factor*. This step relies both on the structure of the grammar and the domain's characteristics associated with the feature list. The definition of pruning factors is given in Chapter 5.1.4. Pruning factors depend on another calculation, the *precedence relation*, defined in Chapter 5.1.5. Precedence relations are used to calculate pruning factors in situations where dynamic bounding functions are used. The final stage multiplies the expected frequency of path extensions (state frequency multiplied by pruning factor) by the expected cost of state expansion. In addition to being used to calculate $E(C)$, these measure are also used to calculate the expected number of correct answers that will be found. This is defined in Chapter 5.3.

The general approach to calculating $E(C)$ is shown in Fig. 1.5 on page 12. The generational grammar, IDP_g, is used to determine the statistical distributions for sets of signal data. These distributions, in turn, define the sample sets and the sample set weightings. The sample sets and weightings are used to calculate the statistical properties of groups of low-level domain events. These properties are then combined to determine the expected properties of higher-level results.

Chapter 5 defines several important concepts. One is the concept of a *singularity*. A singularity can be thought of as a fundamental unit of analysis that repeatedly appears in a domain grammar. By first calculating the properties of a domain's singularities, it is possible to accurately and efficiently calculate the related properties of all the elements of the domain. For example, in the vehicle tracking grammars shown in this thesis, a singularity is a data point that occurs at a specific time-location. A vehicle track is not a singularity, since it spans multiple time-locations but a vehicle-location is a singularity. In the vehicle tracking grammar, other singularities include groups, signals, and noise. By calculating the properties of vehicle-locations, the properties of tracks can be determined. Similarly, a singularity in a natural language processing task might be a verb phrase. Verb phrases are used repeatedly in a natural language domain and understanding the properties of verb phrases would be necessary to determine the properties of sentences or paragraphs.

Another important concept used in the analysis tools is that of the *solution nonterminal (SNT)*. An SNT represents a class of final states or full interpretations. SNTs are used throughout this thesis and are necessary for defining a variety of other concepts including noise, missing data, subproblem relationships, and quantitative analysis tools. SNTs are required for quantitative analysis because their formal specification makes it possible to specify the characteristics of a search space relative to them. This supports the calculation of expected lengths of search paths, the number of search paths, etc., that are needed by other computations.

14.4 Formally Specifying the Structure of Search Spaces and Control Algorithms

The *UPC* formalism defined in Chapter 6 provides a representation of a search space that can explain and predict the behavior of a search control mechanism. In the *UPC* representation, the traditional concept of a search space state is extended to include vectors indicating a state's location in a search space relative to final states in terms of the cost and probability of reaching the final state and the final state's expected utility. The *UPC* formalism can be thought of as computational structure based on statistical characteristics of IDP models that can be used to simulate an optimal problem solving strategy based on IDP statistics and a specification of problem solving operators. Using the *UPC* representation, we can construct problem solving systems capable of achieving the levels of performance predicted by quantitative analysis of IDP domain specifications and the *optimal interpretation control strategy*. The optimal interpretation control strategy is defined in Chapter 6.4. This is a necessary component of an analytical framework because it provides a basis for experimental control, comparison and evaluation. For example, in the experiments in Chapter 7, the base-line used in the experimental comparisons is the performance of a problem solver that uses the optimal interpretation control strategy to evaluation operators. As a consequence, domains with different structures can be compared using identical evaluation function based control architectures or these architectures can be varied to compare performance of different problem solvers within a given domain.

The formal definitions of *UPC* vectors appears in Chapter 6.3. This chapter includes numerous definitions of important *UPC* concepts such as Definition 6.3.3 related to the

probability that a path exists from a given state to a particular set of final states, Definition 6.3.1 related to the expected utility of a state, and many more.

The formal definition of the statistically optimal control strategy in Chapter 6.4 is based on a decision theoretic control strategy that selects operators based on statistical information derived from an IDP specification of a domain and problem solver. At each step of problem solving, the optimal control strategy selects the problem solving action that maximizes the cost/benefit ratio among all available operators. Here, cost is the cost of executing the operator and the benefits are the extent to which the action moves the problem solver closer to the termination, which is to fully connect the base space. These concepts are defined and discussed in Chapters 3 and 11.

Additional strategies that can be similarly used as an experimental control are defined and discussed in Appendix A. These additional strategies include *Total Utility Optimality (TUO)*, which ignores cost, *Utility per Unit Cost Optimality (UUCO)*, which maximizes the utility generated per unit cost, and *Minimum Cost (MC)*, which minimizes the cost of problem solving regardless of the utility of the result generated. Any of these strategies could be used in a fashion similar to the optimal strategy used in the experiments in this thesis. Specifically, where appropriate, these strategies could be incorporated into an experimental testbed and the quantitative results from Chapter 5 could be rederived as needed.

Chapter 6.5 demonstrates the *UPC* vector calculations for a grammar. The grammar is shown in Fig. 6.4 on page 126. The chapter demonstrates the effect of phenomena such as noise and missing data on *UPC* values (Chapter 6.5.2) as well as interacting subproblems (Chapter 6.7). The results are summarized in figures such as 6.5.

14.5 Verification Experiments

The IDP/*UPC* framework is validated in a series of experiments. The experimental testbed consists of a *problem instance generator* that uses a generation grammar, IDP_g, to create problem instances and a problem solver specified by an interpretation grammar, IDP_i, to interpret each problem instance. The first set of experiments in Chapter 7 demonstrate the validity of the analysis framework using the statistically optimal control strategy. The initial validation experiments demonstrate that the IDP/*UPC* framework can be used to accurately predict and explain the performance of a sophisticated problem solver. In these experiments, the problem solver's model of the actual problem domain's structure is distorted in a variety of ways. The intent was to investigate the manner in which a problem solver's performance is affected by a deviation from an ideal domain theory. The experiments demonstrate that *accurate predictions can be made in situations where the problem solver has a model of the problem domain that is identical to the actual problem domain and where the problem solver has a model of the problem domain that is significantly different from the actual problem domain.*

The grammars used in these experiments are shown in Figures 7.1 (page 140) and 7.2 (page 140). IDP_g includes rules representing noise and missing data and IDP_iincludes rules representing bounding functions. Two sets of experiments were conducted. The results of the first set are shown in Table 7.1 (page 142), and the second in Table 7.2 (page 145). In all the experiments, the performance predicted by the quantitative analysis tools was consistent with the actual observed results.

Specifically, each experiment involved two distinct grammars, one corresponding to IDP_g and one to IDP_i. A series of 50 problem instances were generated and measurements of the actual cost of interpreting each instance and the number of correct answers found were

gathered. An "experiment" consisted of 100 such trials. (i.e., an experiment consisted of 100 "samples" each of which included 50 problem instances.) In each experiment, the analysis tools correctly predicted the expected problem solving cost and the expected number of correct answers found. (Note: Correct answers are *not* found in situations where one or more of their required component paths are eliminated by bounding functions.)

The first set of 7 experiments included a base-line (Exp 1) and 6 other experiments in which the characteristics of the IDP_g and IDP_i grammars were varied. For example, in experiments 4 and 5 the problem solver used a model of the domain that was different from the domain model used to generate problem instances. In both situations, the analysis tools correctly predicted the problem solver performance. Experiments 4 and 6 demonstrate the effects noise and missing data have on the cost of problem solving. In experiment 4, the generational grammar is skewed to use more rules associated with noise and missing data. The corresponding cost of problem solving rose dramatically, almost 100%. In experiment 6, the generational grammar is skewed to use fewer rules associated with noise and missing data. The corresponding cost of problem solving dropped significantly.

A second set of 10 experiments is summarized in Table 7.2 on page 145. The approach used is similar to that of the first set. Each experiment consists of 100 trials of 50 instances each and measures the actual cost of problem solving and the number of correct answers found. The emphasis of these experiments is on the characteristics of the problem solver and the accuracy of the analysis tools. In these experiments, the problem solver's semantic functions were modified to rate "bad data" (i.e., data from noisy or missing data rules) and "good data" (i.e., data from non-noisy rules) either higher or lower than it would if it had statistically precise domain data. In all cases, the analysis tools accurately predict the problem solver's performance. This includes experiment 17, which is the most realistic. The problem solver in experiment 17 makes errors of all types in evaluating partial results but the analysis tools still accurately predict its performance.

14.6 Unifying the Representation of Meta-Level and Base-Level Processing

The IDP/*UPC* framework extends the traditional notion of a search space to incorporate abstract and approximate states, and the operators that create, modify, and exploit them, in a unified representation including traditional forms of search-based problem solving. Chapter 8 introduces and defines the concepts of *projection spaces* and *projecting* and *mapping* operators. Projection spaces are abstractions of a base search space in which an approximate, (hopefully) less costly version of a problem can be solved. Projection spaces are defined by special meta-level, projecting operators, and the results of problem solving in these spaces are propagated back to the base space by mapping operators. By defining projection spaces and the associated projecting operators, approximate/abstract operators, and mapping operators in terms of the as extensions of the base space, an integrated perspective of both domain and meta-level processing. This supports the analysis of problem solvers that use sophisticated control mechanisms to function incrementally and simultaneously in a continuum of abstraction spaces.

This contribution supports the analysis of the tradeoffs between meta-level processing and domain processing. This is accomplished by first integrating the the operators associated with projection spaces into an IDP_i grammar and then calculating the quantitative performance characteristics of the problem solver using the previously defined techniques based on the *UPC* formalism. Furthermore, the unified representation formally defines the concept of meta-level processing in such a way that its relationship to domain processing is precise and so that it

can be thought of as part of the same search paradigm as domain processing. This allows entirely new forms of experimentation and analysis to be conducted. Specifically, this not only supports the study of the relative costs and benefits of domain and meta-level processing, but it also supports experimentation in which a known control algorithm is applied in an unknown domain in order to determine the structure of that domain.

The extended search formalism is defined in Section 8.2. The new characterization of a search problem is based on the four-tuple $< \mathcal{S}, \Omega, \omega, \Phi >$, where $\mathcal{S}$ is the start symbol; Ω is the *base search space* with associated CVs and operators; Ω corresponds to the traditional notion of a search space; ω is a set of *projections*, or *abstractions*, of the base search space, each with their associated CVs and operators; and Φ] is a set of ***mapping functions*** from projection spaces back to the base search space. A representation of the extended search space appears in Fig. 8.2 on page 151. Chapter 8.3 shows an example of how *UPC* values are calculated for the extended search space.

14.7 Introducing the Concept of Potential

Chapter 9 introduces the concept of *potential* that is used to formally define the long-term benefits associated with a problem solving action. In many cases, the locally optimal control decision will not result in globally optimal problem solving. This occurs in situations where an operator on a search path does more than simply expand a state and extend a search path – where the operator actually *increases the information available to a problem solver regarding the interrelationships between partial solutions.* i.e., the operator increases the understanding of a partial solution's global significance. In these situations, the operator does not have to actually extend a search path in order to move the problem solver closer to termination, rather, the operator may alter the search space in some way that reduces the cost of problem solving (or increases the effectiveness of problem solving efforts) for some other set of operators. We will refer to this property of an operator as its *potential.*

The concept of potential allows the costs and benefits associated with meta-level operators to be directly compared with those associated with domain operators. Potential is calculated using the analytical tools enabled by the IDP and *UPC* formalisms. In general, the potential of an action is calculated by first examining what the long-term goals of the action are and then determining what the cost of problem solving would be if the long-term goals had already been achieved. The formal definition of potential is in Definition 9.0.1.

Formally, the potential of operator op_i applied to state s_n is:
$Pot(op_i, s_n) = FTN_{\forall S'}\ (P(S' \mid s_n),\ Potential(S'),\ cost_g(S', s_n),\ cost_m(S'))$, where each element S' is a state that can be reached from s_n that increases the information available to a problem solver regarding the interrelationships between partial solutions (i.e., a state with potential), $P(S' \mid s_n)$ is the probability of generating S' given s_n, $Potential(S')$ is a measure of the degree to which S' reduces the cost of problem solving, $cost_g(S', s_n)$ is the expected cost of generating S' given s_n, and $cost_m(S')$ is the cost of realizing $Potential(S')$, i.e., the cost of mapping S' back to the base space.

$FTN_{\forall S'}$, the general computation of $Pot(op_i, s_n)$, is $(P(S' \mid s_n) * (Potential(S') - cost_g(S', s_n) - cost_m(S')))$. This is for situations where the cardinality of S' is 1. For situations where an operator, op_i, represents the first step on paths to multiple S'_i, the computation is more complicated. In these situations, the function FTN determines $Pot(op_i, s_n)$ based on the relationships between the states S'_i. These relationships are defined in terms of the paths from s_n to the states S'_i (i.e., the costs $cost_g(S', s_n)$) and the set of states that are affected when

the potential of the states S_i' is mapped back to the base space. Figure 9.2 (page 158) depicts the possible relationships between states S_i'. Figure 9.2 (page 158) represents the possible relationships in terms of search space paths and base space states.

Chapter 9.1 introduces an important methodology for analyzing potential. This methodology involves automatically transforming an IDP grammar, computing properties of the new grammar, and incorporating these results in the analysis of the original grammar. Specifically, in determining the potential of an operator. This technique is also demonstrated in the analysis of the expected cost of problem solving in Chapter 5 and Appendix D.

The grammar transformation technique suggests a methodology for automatically constructing control mechanisms. The methodology would take a base grammar, create new control mechanisms by automatically transforming the grammar, then evaluate the properties of the transformed grammar. Such a methodology could be used to identify potential candidates for control mechanisms or, perhaps, select the best control mechanisms automatically.

An intuitive example and discussion of potential appears in Chapter 9.2. This example illustrates how the potential of an operator is calculated in a complex domain. A discussion of how potential is incorporated into a control function is presented in Chapter 9.3.

Finally, experimental results are presented in Chapter 9.4 that demonstrate the validity of the analysis tools applied to a problem solver that incorporates the concept of potential in its control strategy. In these experiments, conditional probabilities and values for $\Omega(s)$ for all elements of the grammar are computed a priori. The problem solver computes expected credibilities, costs, and probabilities dynamically, using the previously computed values for conditional probability and Ω. The results of the experiments are summarized in Table 9.1 on page 170. The grammar used in these experiments is modified from the previous experiment to simulate a vehicle tracking grammar. The modified grammar is shown in Fig. 9.11 on page 171. Problem instances from the modified grammar are much more costly to solve than those in the original experiments and this is shown in the experimental results table. The results from two of the original experiments are included for comparison.

The conclusion of the experiments conducted in this chapter are that the analysis tools accurately predict the performance of a problem solver that uses the concept of potential in its control mechanism. The most striking effect of the use of potential is in the increased time required to conduct an experiment. Given that the rating of every problem solving action requires costly computations, i.e., the computation of C_{F_i}, this result was not surprising. Furthermore, we have found that the cost of the computation of $C(S'_{PS})$ can be reduced significantly through the use of dynamic programming techniques that cache intermediate values and reuse them when necessary. Another result is that the use of potential substantially reduces the cost of problem solving. This is consistent with our intuitions that bounding functions reduce the expected cost of problem solving.

14.8 Defining the Pruning, Preconditions, Goal Processing, and Approximate Processing Sophisticated Mechanisms

Chapter 10 introduces the IDP approach to representing a variety of sophisticated control mechanisms in a heuristic problem solver. This chapter and Chapter 11 demonstrate how the IDP/*UPC* framework can be used to analyze sophisticated control mechanisms in real-world domains. The control mechanisms that are modeled include preconditions, goal processing, bounding functions, and approximate processing mechanisms.

Chapter 10.1 presents the details of representing precondition functions in an IDP grammar. Figure 10.2 graphically depicts the control strategy associated with preconditions. To clarify the use of preconditions, consider the production rule: $A \rightarrow B\ C\ D$. Given a control strategy where there are no preconditions, if the problem solver has generated a search state "B," the operator associated with the production rule would be eligible for execution whether or not the required states C and D had been generated. In situations where C and D had not been generated, the operator would fail to generate an A. In contrast, using preconditions, the operator would not be eligible for execution until all the required syntactic elements were present.

To represent preconditions using the IDP formalism, we will define a projection space of satisfied precondition states. This space is created by operators of the form: $\mathrm{A}^{op}_{[B,C,D]} \rightarrow$ B C D. Thus, a symbol with a superscript *op* will represent a state created by a successful precondition operator. *op* will correspond to the operator that will map the abstract state back to the base space. For clarity, the subscripted brackets will include the required syntactic elements used to create the precondition state. A production rule of this form will be created for each precondition operator. More formally, we say that the set of projection operators from the base space to the precondition abstraction space, $OP_{(\Omega,\omega_{precondition})}$, is made up of precondition operators.

In addition, a mapping operator from the precondition projection space back to the base space will have the form: $\mathrm{A} \rightarrow \mathrm{A}^{op}_{[B,C,D]}$. This operator will correspond to the original operator except that it can only be applied to the states in the precondition space. The original operator will be replaced by these two rules. Formally, the set of mapping operators from the precondition abstraction space back to the base space is represented as $OP_{(\omega_{precondition},\Omega)} \in \Phi$.

Examples of precondition specification in the vehicle tracking grammar are shown in Figures 10.4 and 10.6.

A form of goal processing, *focus-of-control goal processing*, is defined in Chapter 10.2. Focus-of-control goal processing is a mechanism that allows a problem solver to selectively rerate problem solving operators based on their relationships with other, dynamically changing operators. An example of goal processing is:
G.4.1. $\text{Goal-TL1}_{[t+1]} \rightarrow \text{I-Track1}_{[t]}$.
Operator G.4.1 is used to raise the ratings of operators that will generate data needed to extend a partial I-Track1 forward in time. Other goal processing operators are shown in Fig. 10.8 on page 180.

Like the precondition operators, the goal processors define a projection space. Example mapping operators for this space are shown in Fig. 10.9 on page 181. For example, M.G.5. $\emptyset \rightarrow \text{Goal-TL1}_{[t]}$ is a mapping operator for a goal. The mapping operators are applied to the states of the goal projection space. The $\emptyset$ notation on the LHS of the production rules indicates that no new states are generated in the base space. Instead, the ratings for instantiated operators is modified to reflect the information contained in the goal state. A detailed discussion of the mapping procedure is presented in Chapter 10.2.

Another contribution made in Chapter 10 is that it formally defines a taxonomy of sophisticated control mechanisms based on approximations and abstractions. These definitions support the automatic synthesis of abstraction operators from an IDP grammar specification. In addition, given that these abstractions are represented in the IDP formalism, alternative strategies can be quantitatively evaluated and compared to each other.

Chapter 10.3.1 formally defines *data approximation.* Approximating data into abstract clusters is a strategy that attempts to efficiently apply constraints to aggregations of data and

it is particularly useful in domains with large quantities of noise. An example of a data approximation rule represented in the IDP formalism is:
C.1. $S1^p \rightarrow S1_{[f_1,t]} \ldots S1_{[f_n,t]}$, where $S1^p$ is the result of aggregating the elements $S1_{[f_1,t]} \ldots S1_{[f_n,t]}$.

This chapter also defines how the abstract aggregation is represented on a blackboard as a *cluster* hypothesis. A cluster hypothesis has a range, R, defined as a convex region encompassing all the component locations, an event class $E = \cup$(component event classes), and a precision $= f_p$(size of R, $\mid E \mid$, f_v), where f_v is the variance of the clustered hypotheses' locations within R. Figure 10.16 on page 188 shows examples of data aggregated into clusters and the associated precision measures. The formal clustering algorithm is shown in Fig. 10.17 on page 189.

Two general approximation strategies can be used to generate abstract states; search approximation and knowledge approximation. These strategies are defined and discussed in Chapter 10.3.2. The specific techniques described here are *eliminating corroborating support* and *level hopping*. Both general strategies are based on reducing the amount of the search space that is explored in generating an interpretation. As a consequence, the resulting interpretation is considered an abstract state in a projection space.

An example of an operator based on eliminating corroborating support from rule 19 of the vehicle interpretation grammar is:
ECS.19 $V1^{ecs}_{[f]} \rightarrow G1_{[f]}$, where the corroborating support from G3 and G7 is eliminated. The original grammar rule is: 19. $V1_{[f,t,x,y]} \rightarrow G1_{[f,t,x,y]}\ G3_{[f,t,x,y]}\ G7_{[f,t,x,y]}$. The superscript of the approximate rule indicates that the rule defines a projection space *ecs*. To be precise, this operators defines the set $OP_{(\Omega,\omega_{ecs})}$. In ECS.19, the corroborating support from G3 and G7 is eliminated.

An example of a IDP level-hopping approximation of rule 19 is:
LH.19 $V1^{lh}_{[f]} \rightarrow S1_{[f]}\ S2_{[f]}\ S5_{[f]}\ S7_{[f]}\ S11_{[f]}\ S15_{[f]}$. The superscript indicates that this rule defines a projection space *lh* and the set $OP_{(\Omega,\omega_{elh})}$. This operator is constructed by fully expanding the group level results that would be used to generate V1 and V2 results in the base space.

Chapter 10.3.3 Discusses an important consideration as to whether or not the abstractions generated by aggregating data, approximating search, and approximating knowledge can somehow be used to reduce the cost of problem solving. Using approximations to developing a general understanding of a particular problem instance and then using that understanding to more effectively control problem solving can be a very effective problem solving technique. An example of an IDP rule for problem solving in an abstract space is:
AP.4 I-Track^{p}1 $\rightarrow$ I-Track1$^p_{[}t]$
T1$^p_{[}t-1]$. This is an abstraction of rule 4 from the vehicle tracking grammar,
4. I-Track1$_{[f,t,x,y]} \rightarrow$ I-Track1$_{[f,t+1,x+vel+acc,y+vel+acc]}$ T1$_{[f,t,x,y]}$.

Chapter 10.3.4 defines the mapping strategies that are used to propagate the results of abstract processing back to the base space. These include the simplest form of mapping, *rating modification mapping*, which uses the results of approximate processing to modify the ratings of operators in order to bias, or focus, the problem solver's actions toward a particular set of search paths. A more drastic alternative is *grammar transformation mapping*. In this approach, the mapping function alters the structure of the problem solving grammar by using the characteristics of the meta-level results to eliminate rules. A third alternative is *explicit plan mapping*. Here, the mapping operator explicitly reorders the sequence of operators on the queue. This is in contrast to *implicit plan mapping* in which the mapping operator adjusts the

UPC values associated with an operator and then allows the queue reordering to be conducted implicitly with the established rating function.

14.9 Demonstrating the Analysis of Heuristic Control

The experiments shown and discussed in Chapter 11 demonstrate that the analysis framework can be applied to heuristic problem solvers in addition to the optimal control strategy used in previous experiments. The experiments in this chapter involve the use of a sophisticated, search-based, blackboard problem solver in a vehicle tracking domain. The IDP domain model used is the complex, real-world domain introduced in the vehicle tracking grammar in Chapter 4. The problem solver used in these experiments included the precondition, goal processing, and pruning control mechanisms defined in Chapter 10. The representation and the analysis associated with these mechanisms is introduced in Chapter 8 and involves expanding the dimensionality of the base search space to include the necessary projection spaces. The operators for projecting base processing to these spaces, for solving problems in these abstract spaces, and for mapping the results back to the base space are all integrated into the IDP formalism.

The problem domain used in these experiments is based on the vehicle tracking domain described in previous chapters and summarized with the generational grammar shown in Fig. 11.1 on page 197. Two general forms of this grammar are used in the experiments. Grammar VT1, which is equivalent to that shown in the figure, models a domain in which there are multiple independent ("I"), pattern ("P"), and ghost ("G") tracks. The actual grammar used in the experiments is also modified so that it creates paths of length 6. In general, it is not feasible or realistic to model domains where phenomena are allowed to represent unrestricted time sequences. The amount of data in a realistic domain would be overwhelming. Instead, real-world domains must employ some form of time-slicing in which data from one given period is analyzed before proceeding to the next time period.

The structure of the heuristic problem solver, including the specific details regarding the implementation of the sophisticated control mechanisms, is presented in Chapter 11.2. The experimental problem solver used a blackboard control strategy based on six levels of abstraction: signal events, group events, vehicle events, vehicle tracks, pattern tracks, and solutions. The evaluation function used is based on the *rating* (i.e., credibility) assigned to a partial result and its level on the blackboard:
$R(op_i(n_j)) = LEVEL(n_j) * CRED(n_j)$, where $R(op_i)$ is the rating for the potential problem solving operator op_i applied to search state n_j, $LEVEL(n_j)$ is the level of the blackboard corresponding to state n_j, and $CRED(n_j)$ is n_j's credibility.

The results of the experiments are described in Chapter 11.3 and summarized in Table 11.5 on page 209. Four sets of experiments were conducted. In the first set, the problem solving operators corresponded to rules of the IDP_g grammar shown in Fig. 11.1 on page 197. In the second set of experiments, pruning operators were added to the grammar. The third set of experiments used goal processing operators and did not use pruning operators. The fourth set used both pruning and goal processing. A series of 50 problem instances were generated and measurements of the actual cost of interpreting each instance and the number of correct answers found were gathered. An "experiment" consisted of 100 such trials. In each experiment, the analysis tools correctly predicted the expected problem solving cost and the expected number of correct answers found.

In general, these experiments indicate that the analysis framework, which was originally based on the use of statistically optimal control decisions, can be extended to incorporate the use of heuristic control functions if the analysis can be tied to statistical characteristics of the domain. These experiments further demonstrate that the framework can be used for analyzing not only heuristic evaluation control mechanisms, but also heuristic control decisions that incorporate other types of control functions such as pruning, goal processing, and preconditions.

Specifically, the experiments show that preconditions are the most effective form of control. They eliminate a great deal of redundant processing and processing that will only lead to failed paths due to the lack of required data. The experiments also indicate that the heuristic evaluation function is a "reasonable" approximation of a statistically optimal control strategy in that the performance of the heuristic problem solver was only 10% worse than optimal. The experiments also demonstrate that dynamic pruning, though not as effective at reducing cost as static pruning, eliminate far fewer correct answers.

14.10 Defining Design Methodologies

Chapter 12 discusses some of the implications and uses of the IDP/*UPC* framework and demonstrates several prototype design tools/theories. Chapter 12.1 introduces two general analysis techniques referred to these as *comparative cost analysis* and *constraint flow analysis*. Constraint flow analysis is used to generate potential sophisticated control mechanisms and is based on determining a directed graph of constraint flow based on the rules of an IDP grammar. Comparative cost analysis is used to conduct pairwise evaluations of alternative control mechanisms. Chapters 12.2 and 12.3 present an extended example of how the IDP formalism is used in the design and analysis of approximate processing techniques.

Chapter 12.4 discusses some of the implications and uses of the IDP/*UPC* framework and demonstrates several prototype design tools/theories not necessarily related to approximate processing. These include the concept of *marker* and *differentiator* that are based on statistical characteristics of a domain derived from an IDP specification. The concept of a marker defines phenomena that are indicative of a high-level event and that can be used to derive top-down control mechanisms that use information about markers to guide search for low-level results. In contrast, the concept of a differentiator defines low-level phenomena that are indicative of specific high-level results. Thus, the presence of certain differentiators can be used to guide problem solving by focusing on a small set of search paths or by eliminating alternative paths. Also, the concept of a differentiator suggests that the IDP/*UPC* framework can be extended to the analysis of interpretation problem solvers that rely on differential diagnosis strategies.

Other design methodologies are introduced for dealing with the *texture* of a problem domain in terms of a domain component structure, or *arity*, (Chapter 12.7), and its utility structure (Chapter 12.8). The importance of arity is that the basic component structure of a problem domain can be exploited to reduce the cost of by ordering problem solving activities to limit the branching factor of the search. The utility texture of a domain can be analyzed using *credibility maps* to estimate the effects of certain kinds of bounding functions.

The developments in Chapter 12 represent prototype design theories and principles for the construction of sophisticated problem solving systems. For example, by relating the specification of approximations to an IDP grammar, a mechanism for automatically generating approximations is introduced. As discussed in Chapter 9, this thesis demonstrates how grammar transformations can be automated and then quantitatively analyzed. Though no experiments or demonstrations were made of how this methodology can be used to automate

the construction of approximate processing mechanisms, the implication is clear. Further, relating approximations to IDP representations allows the development of design theories and principles, such as constraint flow analysis, that can be used in the construction of sophisticated control mechanisms that are based on approximations and abstractions. In addition, Chapter 10 and Appendix C define important concepts and principles for a general, blackboard-based architecture for approximate processing that is, in large part, derived from the IDP/*UPC* framework. Some of the definitions include the concept of *precision* for defining a dimensional index for referencing abstract projection spaces, the four-valued belief system, and other general architectural concepts.

14.11 Experiments with Approximate Processing

Chapter 13 discusses experiments that were conducted to determine whether or not the analysis framework, in its current state of development, can be applied to domains that use sophisticated control techniques based on approximate processing. Specifically, we wanted to test the framework in a domain where the problem solver's control strategy included all forms of approximate processing discussed in Chapter 10.3.

The problem solver used for these experiments is a simplified vehicle tracking grammar based on the grammar introduced in Chapter 12 and shown in Fig. 13.1 on page 253. This grammar is somewhat simpler than the grammar used in the experiments in Chapter 11, but it contains many of the same complexities such as noise/missing data and context sensitive attributes linked through the feature list convention. Three different kinds of operators were used, a level-hopping operator that approximated domain knowledge, a clustering operator that aggregated data in abstract states, and approximate processing operators that generated interpretations in a projection space.

Four experiments were conducted. The first, AE1 (Approximate Experiment 1) did not use any approximate processing. Experiments AE2, AE3, and AE4 used level-hopping, clustering, and approximate processing. The cost of the approximate operators and the mapping functions was varied from 10, to 1, to 50 in the experiments. The results of the experiments are summarized in Table 13.1 on page 255.

The mapping procedure used in these experiments was the transformation strategy described in Chapter 10.3.4. Given a set of input data, the problem solver first uses level-hopping to generate abstract, vehicle level results. These results are then clustered into an abstract states that are processed as using operators derived from base space operators. The results of this processing are mapped back to the base space by using the results of approximate processing to transform the base space grammar.

Chapter 13.1 discusses the modifications that were made to the quantitative tools to support the analysis of approximate processing. These primarily involved modifying the computation of frequency maps and the expected cost of mapping operators.

The experimental results are discussed in Chapter 13.2. A series of 50 problem instances were generated and measurements of the actual cost of interpreting each instance and the number of correct answers found were gathered. An "experiment" consisted of 100 such trials. In each experiment, the analysis tools correctly predicted the expected problem solving cost and the expected number of correct answers found. In all the experiments, the performance predicted by the quantitative analysis tools was consistent with the actual observed results. This verifies that the analysis tools accurately predict the performance of problem solvers that include sophisticated control mechanisms based on abstractions and approximations.

14.12 Future Directions

There are a number of open issues related to the concepts discussed in this thesis. One of the issues is related to the *cost* of computability. The techniques for computing potential used in the experiments described in this thesis are too expensive to be used in a real problem solving system. Consequently, the cost of determining *UPC* values in general is too expensive for real problem solving systems. Future research will attempt to address this issue by defining more comprehensive search space structures (which will be referred to as *macro-structures*) that characterize the overall state of problem solving at various points and that can be used to efficiently estimate the effect a precise calculation of potential would have on *UPC* values. Such an estimate would have a level of accuracy that would approximate some form of optimal processing. Defining the macro-structures will involve formalizing the concepts of *control uncertainty* and *solution uncertainty* in terms of the *UPC* formalism.

Using dynamic estimates of control uncertainty and solution uncertainty, it will be possible to address the general issue of, "When is it worthwhile for the control component to use some calculated estimate of potential in its decision making?" In some domains, it may be advantageous to do this for every single search state. In other domains, it may only be necessary to do this for certain states. For example, a domain may be structured in such a way that the problem solver can determine what action to take for certain states a priori. Similarly, the problem solver may be able to determine the best course of action for states with certain characteristics that are determined dynamically. For example, the problem solver may be able to determine a priori that it should always prune states with a credibility lower than some threshold. Intuitively, it is clear that in many situations the decision to include a calculation of potential will need to be based on the current set of states or on dynamically determined properties of the current set of states.

More specifically, this issue will be addressed by defining the concept of ρ that characterizes the degree to which the estimates of *UPC* values used by a problem solver deviate from the values that would be derived from an ideal domain theory, i.e., from values that take precise calculations of potential into consideration. In many ways, ρ is analogous to the correlation between U, P, and C vectors derived from an ideal domain theory and the estimates of these values used by a problem solver, $\hat{U}$, $\hat{P}$, and $\hat{C}$. However, ρ's representation is based on the additional problem solving cost incurred resulting from inaccuracies in the estimation of *UPC* values and is "inverted" compared to correlation measures. Thus, in situations where $\rho = 0$, $\hat{U}$, $\hat{P}$, and $\hat{C}$ are equivalent to U, P, and C and the problem solving will be optimal in the sense that it will be equivalent to problem solving based on *UPC* values derived from the ideal domain theory. In situations where $\rho > 0$, $\hat{U}$, $\hat{P}$, and $\hat{C}$ and U, P, and C are different and problem solving will not be optimal. Over long periods of problem solving, a problem solver with a ρ value that is greater than 0 will incur unneeded costs and/or generate less than optimal utility.

In general, the performance of a problem solver will vary in a way that is inversely proportional to the value of ρ. When ρ is small (i.e., close to 0), the problem solver will perform well. When ρ is large, the problem solver will perform poorly. Furthermore, when the performance of two problem solvers is compared, the problem solver with the higher ρ value will incur additional costs and/or generate less utility.

The concept of ρ is related to the computation of potential (and *UPC* values) because it can be used to analyze the advantages and disadvantages associated with expending resources to obtain a better estimate of *UPC* values. This will involve addressing issues such as, "When should a problem solver improve its estimates of *UPC* values by executing information gathering actions and when should it simply extend paths in the base search space?"

In general, it is expected that there are many techniques for determining when potential is an important consideration for the control decision making process. These techniques are all sensitive to the structure of a search space including the operators that exist, both base-level and meta-operators, and their interactions with other operators.

As more complex real-world domains are examined, it will be necessary to address the issue of developing design theories based on an increasingly sophisticated understanding of problem structures. In particular, through experimentation and analysis, many commonalities between disparate domains are being discovered that seem to indicate that certain control architectures can be used to form a basis for a very powerful and general form of problem solving. Specifically, it is becoming apparent that approximate processing [Lesser and Pavlin, 1988, Decker *et al.*, 1990] and goal processing [Corkill and Lesser, 1981, Corkill *et al.*, 1982, Lesser *et al.*, 1989a, Lesser *et al.*, 1989b] are forms of processing that can be generalized to many real-world domains. This general strategy is one in which a problem solver uses approximate processing operators to gain a comprehensive view of the data it must interpret, then uses this perspective to guide subsequent problem solving. The implications of these observations will be investigated and attempts will be made to discover principles that will lead to the development of design theories supporting the automatic construction of problem solving operators such as approximate processing operators. Hopefully, it will be possible to demonstrate how this use of the IDP and *UPC* formalisms will support the synthesis of new, more flexible control architectures.

APPENDICES

APPENDIX A

GENERAL OBJECTIVE STRATEGIES AND THE *UPC* FORMALISM

This section will define a general taxonomy of objective strategies in terms of the *UPC* model that can be used to analyze alternative problem solving strategies. To accomplish this, the process of *control* will now be defined as *the process of searching for the best operator to apply.* The space searched by the control mechanism is defined by the *UPC* vectors and will be referred to as *operator space.* When an operator is executed, it usually causes other operators to become eligible for execution. These are rated and then added to a pool of available operators. This pool is identical to operator space. In the IDP/*UPC* framework, the search for the best operator is a linear search of operator space. (In the actual implementation, newly rated operators are inserted into a list of operators sorted by their evaluation function ratings. Consequently, the process of search operator space for the best operator is equivalent to selecting the first element of the sorted list.) The basic control cycle used in the IDP/*UPC* framework is reproduced from a previous section in Fig. A.1.

The evaluation of an operator is based on the objective function used by a problem solver. Thus, investigating the properties associated with alternative objective strategies, which we define as the *experimentation(agent objective strategy) analysis paradigm* in Chapter 1.7, involves the use of alternative evaluation functions. For the experiments described in this thesis, the

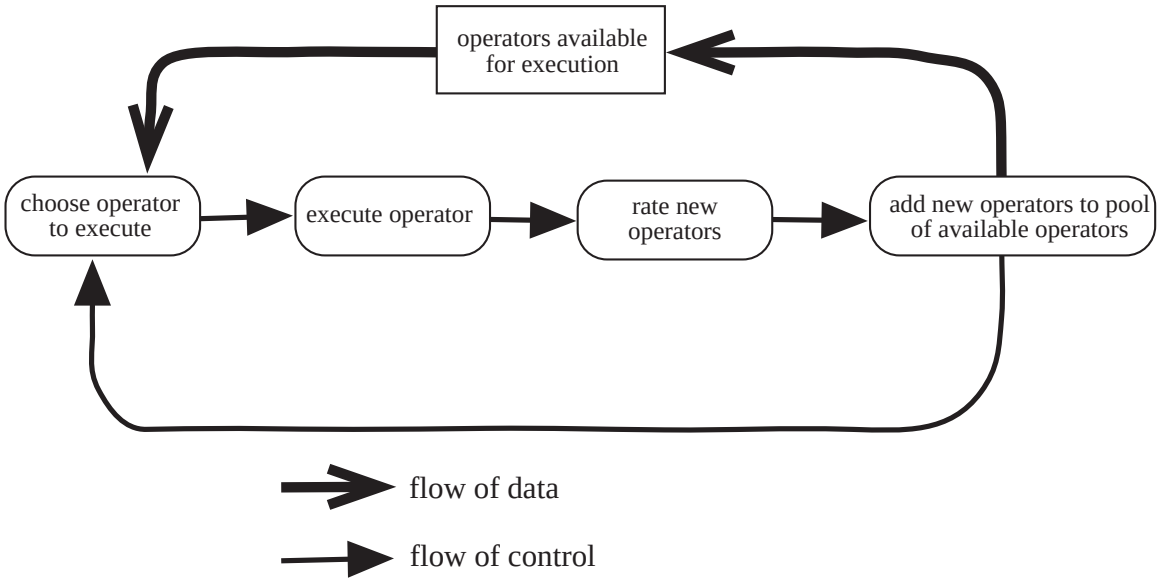


Figure A.1. The Basic Control Cycle

objective strategy used is based on the optimal strategy defined in Chapter 6.4. The following sections specify a taxonomy of objective strategies based on the use of evaluation functions and *UPC* values. This taxonomy can be used in the analysis framework to construct control architectures and simulation tools to investigate the effects of alternative objective strategies. Ultimately, we hope to use this taxonomy to generalize control architectures to new domains and for other general analytical purposes.

A.1 Total Utility Optimality (TUO)

The TUO strategy is to maximize the total amount of utility generated regardless of cost. In the TUO strategy, there is an implication that the problem solver should find the optimal utility as efficiently as possible. (The IDP optimality specification of Chapter 6.4 is essentially the TUO strategy with explicit efficiency requirements.) This strategy can be thought of as always choosing for expansion the path that most efficiently leads to the maximum expected utility. Formally,

Definition A.1.1 *TUO Objective Strategy -* $max_{\forall ni}(s_n(u_i))$, *where* $s_n(u_i)$ *is the expected utility of potential final state* i *that can be reached by extending a path from* s_n. *The problem solver will choose the operator that extends a path leading to the potential final state with the highest expected utility. In the case of ties, the problem solver will choose the path with the lowest expected cost.*

The entries in the *UPC* vectors are *expected values* and that the actual values vary. Consequently, a problem solver will not know the real utility of a final state until it reaches that state and, once a problem solver reaches a final state that is optimal according to the estimated values, it cannot, without additional processing, rule out the possibility that a different final state exists with a higher utility. Given that a domain may exhibit extreme variance, a problem solver must take this fact into consideration. This may require the problem solver to conduct additional search to *prove* that a specific final state has the highest utility.

The TUO strategy is best suited for domains that do not include some consideration of cost when computing the utility generated by the system. More specifically, the TUO strategy is not well suited for domains that require real-time responsiveness.

A.2 Utility per Unit Cost Optimality (UUCO)

This strategy is to pursue a course of action that maximizes the expected utility per unit of cost expended. Formally,

Definition A.2.1 *UUCO Objective Strategy -* $max_{\forall n,i}(\frac{(s_n(u_i)*s_n(p_i))}{s_n(c_i)})$, *where* $s_n(p_i)$ *is the expected probability of generating potential final state* i *by extending a path from state* s_n, *and* $s_n(c_i)$ *is the expected cost of the path to potential final state* i. *The problem solver will choose the operator that is expected to generate the most utility per unit cost.*

The UUCO strategy is best suited for domains where the ultimate utility generated by a problem solver is a function of cost. This is especially applicable for domains where time is the primary cost and where there are implicit (or explicit) real-time responsiveness requirements. For example, for a speech understanding system, there are implied requirements that the system respond to spoken input in a timely fashion.

Situations where this strategy is appropriate are sometimes described as *satisficing* problems [Simon, 1969]. For interpretation tasks, satisficing strategies are based on the assumption that a "good" interpretation, i.e., one that is within some ϵ of the optimal (or "correct") interpretation, has a semantic interpretation that is very similar to that of the correct interpretation. As a result, the problem solver's best course of action may be to return a good answer in a timely manner, rather than an optimal answer requiring significantly more resources to derive.

The UUCO strategy is not well suited for domains where processing costs are irrelevant or where it is imperative that the highest quality solution be found.

A.3 Minimum Cost (MC)

The MC strategy is to minimize the total cost of reaching a final state.

Definition A.3.1 *MC Objective Strategy - $max_{\forall n,i}(s_n(c_i))$. The problem solver will choose the operator that extends the search path with the least expected cost that leads to any final state.*

This strategy can be used when any final state is acceptable, especially for domains where real-time responsiveness is required.

APPENDIX B

MAPPING STRATEGIES

In the experiments that are described in this thesis, and in the examples and related discussion, the mapping strategies that are used do not create any new states in the base search space. They only update *UPC* values based on the information derived from abstract states in projection search spaces. This, however, is not a requirement of the IDP/*UPC* framework, it is merely the convention that has been used for this thesis. The following is a taxonomy of classes of mapping strategies that we have identified and that can be incorporated in the IDP/*UPC* analysis framework in ways that are consistent with the definitions presented in this thesis.

Sophisticated Control - The most straight forward form of mapping is when the problem solver uses the results of search in projection search spaces to increase the accuracy of the *UPC* estimates in the base search space and thereby increase the efficiency of problem solving. In this method, states in the base search space are linked to states in a projection space and the mapping function uses information from states in the projection search space to modify *UPC* values in the base space. Problem solving in the base space is then directed by a standard control architecture, such as TUO.

As described previously, this is the mapping strategy used throughout this thesis. The process of mapping a state in a projection space to states in the base space is represented by meta-operators of an IDP grammar with an abstract state as a RHS of a base space state. The base space state on the LHS of the rule defining the meta-operator defines the information that is used to update *UPC* values, and the component set of the abstract state on the RHS of the rule defines the states that have their *UPC* values modified.

Refinement - An alternative approach is to modify the control architecture to include operators that map states in a projection space back to newly created states in the base space. In general, these operators are very costly and can be thought of as collections of operators from the base space. (As described previously, the semantic functions corresponding to the base space operators must be executed to generate an interpretation.) Consequently, refinement can also be thought of as transforming a search problem to a *sequencing problem*. This method essentially transforms the abstract solution into a *plan* for solving the base search problem. Such a plan would have well defined tasks that correspond to base space operators as well as mechanisms for verifying that the plan was working.

Constraint Directed Search - A hybrid approach involves treating the projection space solution as a constraint network, such as the constraint networks defined by Fox [Fox, 1983]. Selective refinement is used to map portions of the projection space solution back to the base search space, or sophisticated control based search can be used to achieve a similar result. The partial result that is generated in the base space is then projected back to the abstract space and incorporated into the abstract solution. The constraints implied by the more precise partial result from the base space are then propagated to the components of the abstract solution. This process can continue until a solution is determined.

Approximate Processing (Satisficing) - Another direct method for using the results of projection space search is to return them as the actual result of problem solving. This method is applicable for domains that admit satisficing solutions [Simon, 1969]. In particular, approximate processing can be used when an acceptable solution can sacrifice precision, certainty or completeness [Lesser and Pavlin, 1988, Lesser *et al.*, 1988b, Decker *et al.*, 1990].

APPENDIX C

IMPLEMENTING AN APPROXIMATE PROCESSING SYSTEM

A system that uses the approximate reasoning methods described in this thesis must address a number of general issues. These issues are related to how approximate processing affects the system's representation of belief and uncertainty, its representation of input data and partial results, its organization of knowledge, and its control of problem solving.

In the state space paradigm, a problem solver assigns a *rating* to each partial result that it uses to determine which of them to extend next. This rating is the system's best estimate of the degree to which an action reduces the distance to termination. In an optimal system, the correlation between the system's rating and the actual best rating (denoted P_a or $P_a(PR_i)$ where PR_i represents "Partial Result number i") is 1, and at each point the problem solver can optimize its performance by expanding the partial result with the highest rating. Thus, there are two forms of uncertainty associated with problem solving. The first is simply the probability that a given partial result is on a correct solution path, $P_a(PR_i)$. This reflects uncertainty inherent in a given domain. The second is the correlation between the system generated rating and $P_a(PR_i)$ (denoted $C(r, P_a)$ or $C(r, P_a(PR_i))$). Approximate processing affects a system's representation of both forms of uncertainty.

Approximate search that eliminates corroborating support can generate partial results that would not have been generated by exact search. These states will have a nonzero rating because constraints that would otherwise reduce their ratings to zero have not been applied. Furthermore, the generation of correct partial results will also be underconstrained and the problem solver will not be able to assign them accurate ratings. As a consequence, $C(r, P_a)$ will be reduced for states generated by approximate search.

Similarly, if a problem solver eliminates competing alternatives, it does so by implicitly assigning them a rating of zero. This has no effect if the partial result is not on any solution path, but reduces $C(r, P_a)$ if it is. In addition, premature pruning of potential solution paths could restrict the problem solver's ability to generate accurate ratings for other states and, indirectly, reduce $C(r, P_a)$.

Data approximations are aggregations of individual partial results, PR_i, each associated with a corresponding $P_a(PR_i)$. The probability of the new, abstract state being on a solution path is a function of the individual PR_i's that varies within and between domains. There is no guarantee that, for a given domain, PR_i can be estimated accurately and, therefore, it is very likely that use of data approximations will reduce $C(r, P_a)$.

The use of approximate knowledge has an effect similar to that of approximate search. Underconstrained operators generate partial results that would not have been generated otherwise and whose ratings are overestimates. They also generate correct, but over rated, partial results. Both of these events result in a reduction of $C(r, P_a)$.

The use of approximate processing affects a system's representation of partial results and solutions. In a state space search, a partial result or a solution is represented as a sequence of operators that were applied to the start state. As discussed in [Decker *et al.*, 1990], the use

of approximate search and approximate knowledge result in partial results and solutions that are offset along the axes defining solution quality. In addition, the use of approximate data confounds the solution path by extending it over multiple levels of abstraction. These two phenomena extend the dimensionality of a partial result or a solution to represent where in the state-space a given state exists in terms of solution quality and data abstraction.

Approximate processing affects the way a problem solver organizes its knowledge. There may be a large, possibly infinite number of data abstraction levels in any given domain. In addition, the ranges along the axes defining the quality of partial results and solutions may be continuous. As a consequence, it may not be feasible for a problem solver to decompose its knowledge into a discrete number of operators and some other form of organization may be required.

Finally, approximate processing affects the way a problem solver applies its knowledge. In many problem solving systems, resource constraints, such as time constraints, are implicit. Other systems are required to incorporate resource constraints into their reasoning. In order to exploit approximate processing techniques successfully, a problem solver will need to recognize situations where the risk reward ratio makes it advantageous to use approximate processing. This is important because data approximation may yield levels of abstraction where aggregations of states are so large they are meaningless, or approximate search and approximate knowledge may generate solutions of undesirable quality. In these situations, resources expended on approximate processing were wasted since no usable results were produced. The underconstrained nature of approximate search and knowledge approximation can lead to the creation of a large number of partial results. If the cost of processing additional partial results is greater than the saving gained using approximate processing, resources will again have been wasted. Finally, if the cost of generating abstract state spaces is higher than the corresponding saving, resources will again have been wasted. Furthermore, a problem solver will have to be capable of refining approximate results if additional resources become available.

The following sections discuss how these issues were addressed in the experimental problem solver described in Chapter 13 that uses approximate processing.

C.1 Defining Projection Spaces

In Chapter 10.3.1 we define precision and discuss how it it used to represent a level of abstraction. This definition is used extensively in the following sections.

C.2 Belief Representation and Uncertainty

Uncertainty arises in any system from the *reliability* of the initial data, the *imprecision* of that data or the language used to represent it, the *incompleteness* of the data, and the *aggregation* of the data from data approximations or multiple sources[Bonissone and Decker, 1986]. Additional uncertainty is introduced with the use of approximate search, data and knowledge. For example, the credibility of a partial result constructed by ignoring potentially corroborating data must be distinguishable from the credibility of a similar partial result constructed using all available data. In the first case, further processing could raise the credibility of the partial result, but in the second case, no amount of processing will increase the credibility of the partial result because all corroborating data has already been considered.

In addition, our system is capable of making certain *assumptions* about the state of problem solving in order to develop various approximations. For example, the system can *assume* that a

required piece of data is available in order to make a best/worst case assessment of a potential solution path.

To represent uncertainty caused by the use of approximate search, data, and knowledge, the representation of credibility must be expanded to a four-valued system. The new belief system was derived from evidential reasoning [Lowrance and Garvey, 1982] and is similar to that in RUM [Bonissone *et al.*, 1987]. The belief in a hypothesis is now represented by a measure of positive belief (certainty) and of negative belief (refutation). To represent the completeness of the solution, the positive and negative beliefs are further divided into upper and lower bounds. Belief in a hypotheses therefore is summarized with four values, *(certainty, plausibility, refutation, doubt)*, where:

certainty - The lower bound of belief in a partial result. This is a measure of the amount of irrefutable evidence supporting the partial result.

plausibility - The upper bound of belief in a partial result. This is a measure of the degree to which available evidence does not refute the partial result. No amount of consistent processing will raise a partial result's certainty above its plausibility.

refutation - The lower bound of disbelief associated with a partial result. This value is analogous to certainty.

doubt - The upper bound of disbelief associated with a partial result. Similar to plausibility, doubt reflects the degree to which available evidence does not support a partial result. No amount of consistent processing will raise a partial result's refutation above its doubt.

The need for four values can be demonstrated with a simple example. Approximate search is implemented by allowing a knowledge source to make an *assumption* that some relevant supporting data exists. After this assumption is made, the system needs to represent *certainty* so it can determine if an approximate solution meets the satisficing criteria. However, since some supporting data is not used, *certainty* will be less than it would have been using exact processing. Therefore, *plausibility* needs to be represented so that the problem solver does not prematurely eliminate from consideration a result that could be improved with additional processing. Furthermore, since problem solving control is based partially on the credibility associated with partial results, it is necessary to temper overly optimistic assumptions by representing the associated *doubt* that the assumption might be false. Finally, it is necessary to represent *refutation* in order to differentiate assumptions that are found to be false from assumptions that are merely unconfirmed. Figure C.1 introduces a graphic representation of the four-valued belief system.

Furthermore, second-order relationships between the elements of the belief system can also be used to control problem solving. Specifically, various measures of *ignorance*[Lowrance and Garvey, 1982] and conflict can be computed. Ignorance measures, such as $(plausibility - certainty)$, $(doubt - refutation)$, $(1 - plausibility)$, $(1 - doubt)$ and $(1 - (certainty + refutation))$, indicate the amount of useful work that can be done to refine the belief in a hypothesis. Conflict measures, such as $(1 - (certainty + refutation))$, indicate the consistency of the processing that generated a hypothesis.

By changing the belief representation used by the problem solver, we are trying to enable the system to reason about the control of uncertainty. Specifically, we want the system to be able to reason explicitly about the tradeoffs between the certainty, completeness and precision

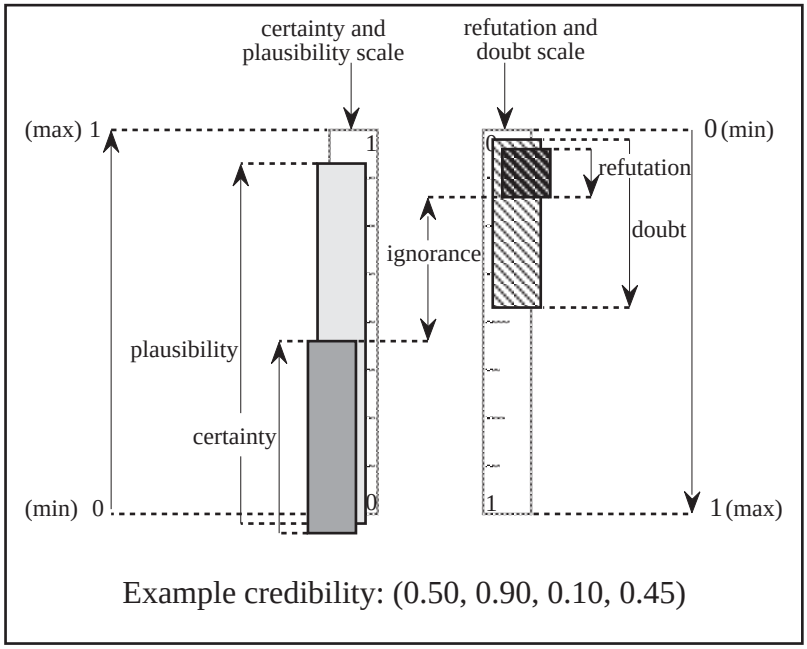


Figure C.1. Graphic Belief Representation Key

of a partial result. In order to accomplish this, we are also enabling the the system to represent not only belief in a partial result, but also the ignorance (i.e., noncommitment) and refutation. These changes will allow the system to reason about the following kinds of uncertainty.

- Ambiguity and uncertainty inherent in the grammar.
- Sensor errors that are correlated to true data.
- Sensor errors that are inherent in the domain.
- Sensors that span ranges.
- Errors that are correlated to other phenomena (masking).
- Uncertainty caused by aggregating data.
- Uncertainty introduced by approximate reasoning mechanisms.
- Uncertainty introduced by approximating goal states.
- Uncertainty resulting from incomplete processing.

This belief system allows the problem solver to represent a number of interesting and useful states of uncertainty. Graphic examples of these states are shown in Fig. C.2.

Complete Certainty - The problem solver can represent its unequivocal certainty that an event occurred by assigning the corresponding partial result the belief $(1, 1, 0, 0)$.

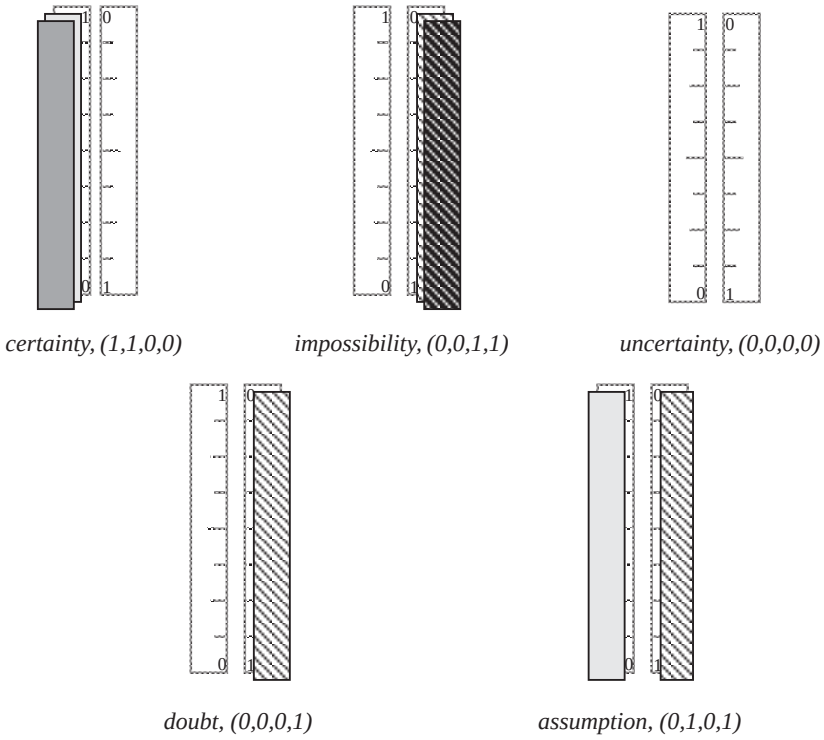


Figure C.2. Belief Examples

Impossibility - The problem solver can represent its certainty that an event did not occur by assigning the corresponding partial result the belief $(0, 0, 1, 1)$.

Complete Uncertainty - The problem solver can represent its complete lack of conviction regarding an event with the belief $(0, 0, 0, 0)$.

Assumption - With the new belief system the problem solver can pursue a line of default or conjectural reasoning by assigning a hypothesis the belief $(0, 1, 0, 1)$.

Doubt - The problem solver can represent circumstantial refutation of an event with the belief $(0, 0, 0, 1)$. For example, if a knowledge source attempts to locate certain data and is unable to do so, it might want to consider the corresponding hypothesis doubtful.

In addition, various relationships between the belief elements indicate the state of ignorance associated with a partial result.

- $(certainty - plausibility)$ and $(refutation - doubt)$ indicate the amount of ignorance that can be resolved with local processing.
- $(1 - plausibility)$ and $(1 - doubt)$ indicate the amount of ignorance that can not be resolved locally.
- $(1 - (certainty + refutation))$ indicates the total amount of ignorance potentially resolvable locally or externally.

- $((plausibility + doubt) - 1)$ indicates the amount of conflict associated with a partial result.

C.3 Modifying the Problem Solving Architecture

In order to extend the problem solver to support approximate processing, significant changes were made to the problem-solving architecture. The resulting architecture allows a range of approximate problem-solving strategies to be efficiently integrated. The implementation of these extensions required that the following questions be resolved.

1. How should intermediate results of problem solving be represented so that precise and approximate intermediate results can be combined in further processing?
2. How should a knowledge source be structured so that it can exploit intermediate results of varying levels of approximation?
3. How should approximate knowledge sources of different types be organized and controlled?
4. How should uncertainty caused by the use of approximate search, data, and knowledge be represented?
5. How should the control architecture be structured so that it can smoothly integrate different types of approximate processing strategies?
6. How should the control architecture be implemented to minimize overhead when the current approximate processing strategy uses only a subset of the partial results and a subset of the applicable knowledge?

C.3.1 Data Representation

Data approximations are created by expanding the representation of precise data along one or more dimensions. The semantics associated with approximate data should be consistent with those associated with precise data so that meaningful combinations of the two can be constructed. If approximate data has a significantly different semantic interpretation than precise data, it might not be possible to incorporate both into a single partial result and this might severely limit any advantages offered by approximate processing.

For example, a single valued location attribute can be expanded to cover a range of locations. Several acoustic signals might be combined, or *clustered*, into a single partial result that encompasses not only the area of the sensed data, but areas where no data was sensed as well. The new data cluster has several interpretations. It can take on an *existential* interpretation, such as "there is some support for a signal source *somewhere* in this area", or it can take on a *universal* interpretation, such as "there is some support for a signal source at *every* point in this area." If some knowledge source activity involves checking for physical overlap among data, then the existential and universal interpretations will produce distinct results. In particular, determining whether or not an existentially interpreted datum and a universally interpreted datum overlap might be prohibitively expensive or impossible. On the other hand, if precise and approximate data are readily interchangeable, then a system has tremendous flexibility in

deciding when to produce approximate data, how to incorporate it into the solution, and how much of it to use.

In order to combine both precise and approximate intermediate results, the vehicle tracking problem solver was modified to represent *all* partial results as clusters of data with characteristics defined by ranges of values. Specifically, each partial result is now represented by a set of event classes, E, and a location range, R. In addition, partial result attributes were extended with the addition of a domain specific *precision*, or "level of approximation", statistic. A partial result's precision is a function of the size of R, the size of E, and the variance of the partial result's likely location within R. This extension effectively expands the dimensionality of the blackboard along an axis corresponding to precision.

Using this representation, an existential interpretation of data has been adopted. An *exact* hypothesis is represented with a range attribute specified by a single point, an event class set with cardinality one, and a precision of zero[1]. A *cluster* hypothesis has a range, R, defined as a convex region encompassing all the component locations, an event class $E = \cup$(component event classes), and a precision $= f_p$(size of R, $\mid E \mid$, f_v), where f_v is the variance of the clustered hypotheses' locations within R.

Figure C.3 shows several examples of approximated data. Part (a) is an exact location hypothesis with precision zero; the size of the range spanned is zero and the vehicle's variance within that range is also zero. The location cluster shown in (b) is less precise than the exact location hypothesis because nine distinct vehicle hypotheses have been aggregated into one. The result is a cluster spanning a larger range and having a non-zero variance within that range. Even though they have similar sized ranges, (b) is more precise than the location cluster shown in (c) because the variance of vehicle locations within (c) is much greater than in (b). (d) highlights the adverse impact wide distributions of data can have on a cluster's precision. Despite having a slightly smaller range,(d) is less precise than either (b) or (c) because of the large variance of vehicle locations it encompasses. (e) shows the adverse effects of clustering multiple event classes. (e) is similar in size and variance to (c) but it is much less precise because it encompasses multiple event classes. (f) and (g) show how quickly precision is reduced when widely distributed data with multiple event classes is clustered.

C.3.2 Knowledge Organization

The introduction of approximate knowledge sources leads to a range of options for organizing a problem solver's knowledge. At one end of the spectrum, there is a specific knowledge source for each form and level of approximation. For example, in a vehicle tracking domain, there could be a specific synthesis knowledge source for eliminating corroborating support of $necessity \leq 0.1$, a specific track extension knowledge source for extending data with precision ≤ 5, and so forth. At the opposite end, there is a single, general knowledge source corresponding to each of the original knowledge sources that encompasses the full range of exact and approximate processing strategies.

The approach taken in modifying the problem solver architecture was at the second end of the spectrum. In order to organize and control the the different types of approximate knowledge sources and to enable a knowledge source to exploit intermediate results of varying levels of approximation, the original knowledge sources were parameterized and restructured so

[1] Because the precision measure is unbound, the problem solver uses an inverted precision scale. Thus, exact data has precision zero, and greater values of precision indicate less precise data.

Precision (ec)(size) + s + s

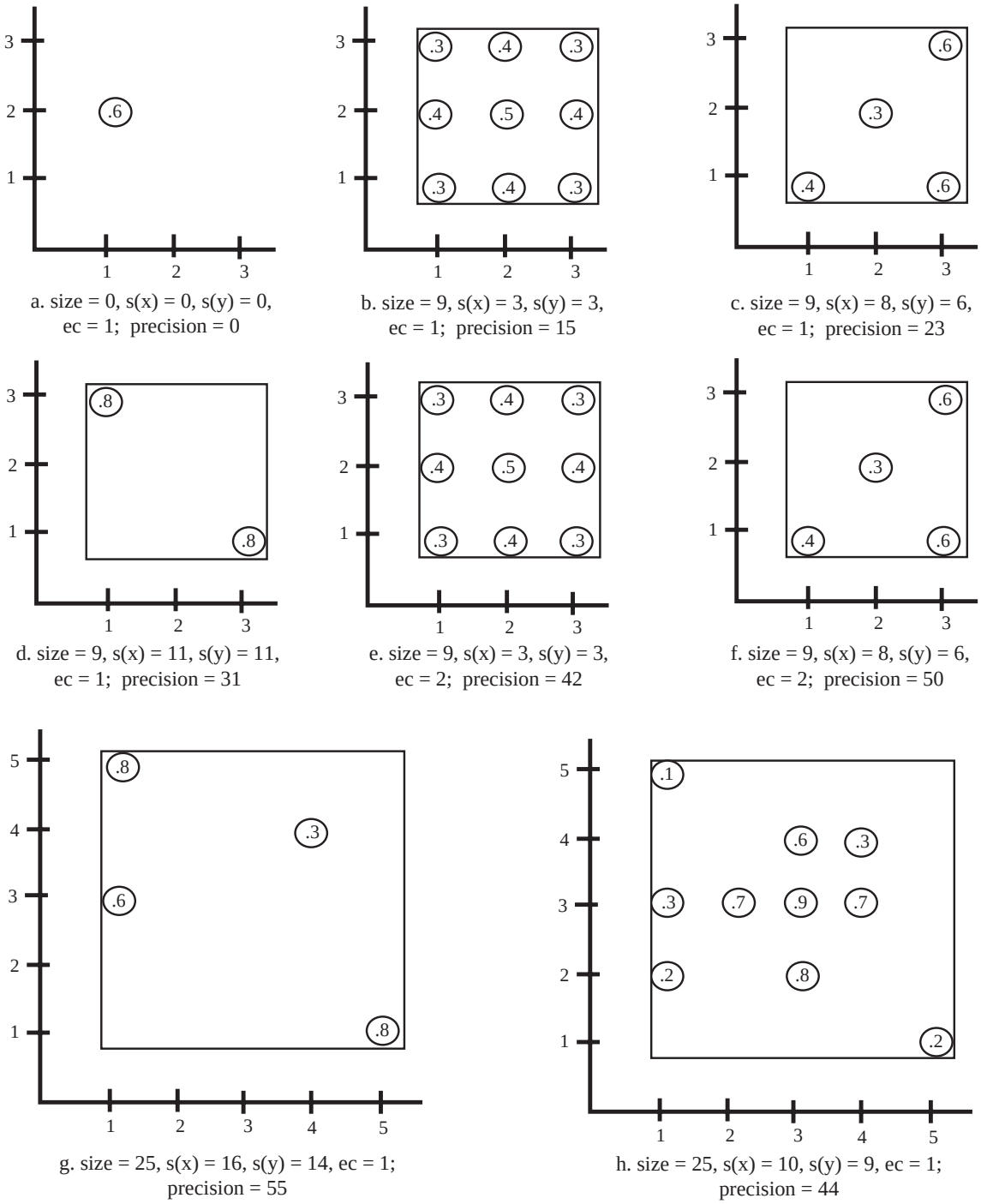


Figure C.3. Examples of Precision Metric

that they are now capable of working with any level of approximate data. Belief combination functions and other domain problem-solving functions were modified to take into account the new interpretation of data described in Chapter C.3.1. For example, synthesis knowledge sources now use the precision statistic to reason about the probability that location hypotheses overlap. Similarly, extension knowledge sources use the precision statistic to reason about the probability of a hypothesis satisfying velocity and acceleration constraints. With these modifications, knowledge sources process exact and approximate data identically and problem solvers can use exact and approximate processing interchangeably.

In addition, a *clustering* knowledge source for aggregating data was added. The precision of the data approximations produced by the clustering knowledge source is controlled by two parameters, C, the number of clusters to form, and I, the *information loss threshold*. To combine individual partial results, the clustering mechanism creates a new hypothesis with characteristics subsuming all of the clustered hypotheses. The clustering mechanism cycles through a set of input data forming clusters until a cluster is generated with a precision $\geq I$ or until the input data has been combined into C or fewer clusters, at which point it halts and outputs the generated clusters.

In order to control the different types of approximation, an *approximation control block* (*ACB*) was added to each knowledge source instantiation. After determining that it needs to use approximate processing to meet some objective, a specific approximation strategy is implemented by adjusting the *ACB*s of the appropriate knowledge source instantiations. When it is invoked, a knowledge source internally chooses from its repertoire of approximations the best way to satisfy the specification defined in the control block. As will be shown later, this method is extremely flexible and allows for a wide range of approximation strategies. The approximation control block contains the following slots.

Input Rating Threshold (IRT)**:** The IRT limits the number of partial results used as input by a knowledge source. A knowledge source ignores all input data with a rating $< IRT$. This approximation is based on a non-global evaluation of the input data and is therefore ill-defined. In situations where a problem solver predicts that it cannot form an acceptable answer using well-defined approximation techniques, it can risk using ill-defined approximations that may lead to incorrect solutions, or no solutions at all.

Output Rating Threshold (ORT)**:** The ORT limits the number of partial results generated as output by a knowledge source. A knowledge source discards all partial results it forms with rating $< ORT$. This approximation is based on a non-global evaluation of the partial results formed by a knowledge source and is therefore ill-defined.

Input Precision Filter (IPF)**:** The IPF indicates the level of approximate data the knowledge source retrieves from the blackboard. A knowledge source will not consider potential input data with precision $> IPF$.

Work Level Precision Filter (WPF)**:** The WPF indicates the level of data approximation the knowledge source actually processes. It has the form (N, C, I), where N = maximum number of partial results to use as input, C = number of clusters to form, and I = the *information loss threshold*. A knowledge source uses these values to cluster input data to the desired level of approximation. By manipulating the WPF, a problem solver can specify the quantity of the highest rated input hypotheses to process, the number of clusters to form, and the amount of precision it is willing to sacrifice.

Output Level Precision Filter (OPF)**:** The OPF determines the level of data approximation of the knowledge source outputs and has the form (N, C, I), where N, C, and I have the previously defined interpretations.

Search Approximation Level (SAL)**:** The SAL indicates the knowledge source's level of approximate search. The SAL has the form (SL), where SL specifies the approximate search strategy to use. Specifically, for synthesis, $SL = n$ indicates that the knowledge source should assume the existence of any subtree of the grammar with $necessity < n$, and for extension, SL indicates the number of time frames it should skip when extending a track.

Knowledge Approximation Level (KAL)**:** The KAL indicates the level of approximate knowledge the knowledge source should use and has the form (KL), where KL specifies which knowledge approximations to use. Different values of KL indicate whether a synthesis knowledge source should ignore constraints or perform level-hopping. Similarly, KL specifies whether or not an extension knowledge source should ignore velocity or acceleration constraints.

By manipulating combinations of parameters in the approximation control block, a problem solver has great flexibility in its use of approximate processing. It can adopt well-defined strategies that are either local or global in extent. For example, it can tailor the precision of a specific area of the solution by setting the appropriate approximation control block parameters in a few specific knowledge source instantiations. Alternatively, it can implement a strategy where all knowledge sources work at a given level of precision, P, by first grouping all data to level P with the clustering knowledge source, then setting every knowledge source's IPF to P. Furthermore, when the situation warrants, a problem solver can also use ill-defined approximation strategies. This can be done with the IRT, the ORT and the N component of the precision filters. Finally, a problem solver can combine thresholding filters, such as the IRT and the ORT, with clustering to moderate the effects of ill-defined approximations.

APPENDIX D

EXAMPLE OF FREQUENCY MAP CALCULATION

This section presents a detailed example of the processes that have been implemented to calculate frequency maps. The grammar that will be used for these examples, VTG-1, is shown in Fig. D.1. This grammar, which is a stylized vehicle tracking grammar, was used for various examples in previous chapters. It is smaller than the full vehicle tracking grammar used in Chapter 11, but it contains many of the same complexities and is preferable for this example because it is smaller and more manageable for the representations that will be used.

As alluded to in Chapter 5.1, an important concept in the calculation of frequency maps is the singularity. A singularity is a data point that occurs at a specific time-location. For example, a vehicle track is not a singularity, since it spans multiple time-locations but a vehicle-location is a singularity. In the vehicle tracking grammar, other singularities include groups, signals, and noise. For a given singularity, a CSS represents the distribution of terminal symbols that can be derived from a singularity. For a vehicle-location or group singularities, the CSS would be the distribution of signal data that can be generated from it.

The calculation of a frequency map for a domain grammar consists of two distinct phases. The first phase is the calculation of a frequency map for the nonterminals corresponding to the

P.1.1.	T	$\rightarrow$ T1 A	p=0.33
P.1.2.	T	$\rightarrow$ T2 B	p=0.33
P.1.3.	T	$\rightarrow$ T3 C	p=0.33
P.2.	T1	$\rightarrow$ A A	p=1.0
P.3.	T2	$\rightarrow$ B B	p=1.0
P.4.	T3	$\rightarrow$ C C	p=1.0
P.5.	A	$\rightarrow$ G1 G2	p=1.0
P.6.	B	$\rightarrow$ G3 G4	p=1.0
P.7.	C	$\rightarrow$ G5 G6	p=1.0
P.8.1.	G1	$\rightarrow$ S1 S2	p=0.9
P.8.2.	G1	$\rightarrow$ S1 S2 S6	p=0.1
P.9.	G2	$\rightarrow$ S11 S12	p=1.0
P.10.1.	G3	$\rightarrow$ S1 S11	p=0.9
P.10.2.	G3	$\rightarrow$ S1 S11 S12	p=0.1
P.11.	G4	$\rightarrow$ S2 S6	p=1.0
P.12.1.	G5	$\rightarrow$ S2 S11	p=0.9
P.12.2.	G5	$\rightarrow$ S1 S2 S11	p=0.1
P.13.	G6	$\rightarrow$ S6 S12	p=1.0

Figure D.1. Full Grammar VTG-1 for Tracking Vehicles Through Multiple Time-Locations

CSS Num	Singularity	Rules	Characteristic Signal Set	Distribution Factor
1.	A	(5, 8.1, 9)	(S1, S2, S11, S12)	0.9
2.	A	(5, 8.2, 9)	(S1, S2, S6, S11, S12)	0.1
3.	B	(6, 10.1, 11)	(S1, S2, S6, S11)	0.9
4.	B	(6, 10.2, 11)	(S1, S2, S6, S11, S12)	0.1
5.	C	(7, 12.1, 13)	(S2, S6, S11, S12)	0.9
6.	C	(7, 12.2, 13)	(S1, S2, S6, S11, S12)	0.1

Figure D.2. Characteristic Signal Sets for VTG-1

highest-level singularities. The second phase involves using cached values from the first phase to calculate the frequencies for the non-singularities in the grammar. These phases, and the constituent parts, are described in the following sections.

D.1 Frequency Map Calculation, Phase I

In phase one, the frequency map of each of the individual singularities is calculated. This is done as generally described in the thesis. First, a set of *characteristic signal sets*, or CSSs, is generated for each singularity. From these, a frequency map is computed. These steps are described in the following sections.

D.1.1 Singularity CSS Calculation

The CSSs for singularities are calculated using the generational grammar, IDP_g. The distribution functions in IDP_g determine the statistical distributions for the CSSs. These distributions define the weightings used to combine the results from different CSSs into a frequency map. This is necessary because it is not sufficient to determine the distributions of *individual* low-level domain events. Instead, it is necessary to determine the distribution of *groups* of low-level events that can be used to generate higher-level interpretations.

Thus, CSS calculations involve two parts. The first part is the calculation of the elements of each possible CSS, the second is the calculation of the statistical distribution of all the CSSs. Figure D.2 shows the CSSs for the singularities in grammar VTG-1. The distributions are calculated in a top-down fashion from VTG-1's ψ functions. In the figure, each row indicates the singularity from which the CSS was derived, the grammar rules used to derive the CSS, the elements in the CSS, and the CSS's distribution factor. Note that each CSS is identified with a singularity. We will refer to these as the *root singularities* because all other singularities are derived from them. For example, the distribution factor of the first CSS in the figure is calculated from the distribution of the rules that were used to generate it. These RHSs were 5. $A \rightarrow G1G2$ with probability = 1; $G1 \rightarrow S1S2$, with probability 0.9; $G2 \rightarrow S11S12$, with probability 1.0. The distribution factor for this CSS is then $1.0 * 0.9 * 1.0 = 0.9$.

D.1.2 Singularity Frequency Map Calculation

Once the CSSs are calculated for each singularity, the frequency map for each CSS is determined. For the CSSs of the singularities in VTG-1, the individual frequency maps are

Sing.	CSS	Dist	A	B	C	G1	G2	G3	G4	G5	G6	S1	S2	S6	S11	S12
A	1	0.9	1			1	1	2		2		1	1		1	1
A	2	0.1	2	2	2	2	1	2	1	2	1	1	1	1	1	1
B	3	0.9		1		2		1	1	2		1	1	1	1	
B	4	0.1	2	2	2	2	1	2	1	2	1	1	1	1	1	1
C	5	0.9			1		1		1	1	1		1	1	1	1
C	6	0.1	2	2	2	2	1	2	1	2	1	1	1	1	1	1

Figure D.3. CSS Frequency Map for Singularities

	A	B	C
A	1.1	0.2	0.2
B	0.2	1.1	0.2
C	0.2	0.2	1.1

Figure D.4. Frequency Map for Root Singularities in Grammar VTG-1

shown in Fig. D.3. Each row of the figure contains the name of a singularity, a specific CSS for the singularity, the distribution factor for the specific CSS, and the frequency map for the CSS. The individual frequency elements in the map are calculated for each element, n, of the interpretation grammar IDP$_i$. For a given production rule, p, for which n is the left-hand-side element, n's frequency is calculated in a two step process. First, the frequency of n from each of the RHSs of p is calculated as the product of the frequencies of the elements of the RHSs, or $\prod_j$ Frequency(e_j), where each e_j is an element of the RHS. Second, the frequencies from each of the RHSs of p are combined by a function that is specific for n. For singularities, the combination function is "addition." Thus, the frequency of n from p is the sum of the frequencies from each of the RHSs of p. In the case where n is a terminal symbol, its frequency corresponds to the number of occurrences in the CSS.

For example, in the frequency maps shown in Fig. D.3, the entry for G3 in the first row in 2 because, given the data in the CSS, there are two rules, 10.1 and 10.2, that can be used to generate distinct G3's. Nothing can be generated with a G3 in the first CSS. In the second row, the frequency of G1 is two because there are two rules, 8.1 and 8.2, that can be used to generate distinct G1's. The two G1's are combined with the single G2 to generate $2 * 1 = 2$ distinct A's.

The CSS frequency maps are combined into the singularity frequency map shown in Fig. D.4. This frequency map is used to calculate the frequencies of non-singularities. It only contains information relevant to root singularities. This is because only root singularities appear directly in the LHS of grammar rules corresponding to non-singularities. The map is formed by multiplying each row of the CSS frequency map by the distribution factor and then summing. Thus, the entry for A in the first row is $(1 * 0.9) + (2 * 0.1)$.

	A	B	C	G1	G2	G3	G4	G5	G6	S1	S2	S6	S11	S12
F_n^B	0.5	0.5	0.5	1.1	0.7	1.1	0.7	1.7	0.4	0.7	1.0	0.7	1.0	0.7

Figure D.5. Domain Singularity Frequency Map for VTG-1

CSS Num	CSS Generation Rules	Characteristic Signal Set	Distribution Factor
1.	(1.1, 2)	(A, A, A)	0.33
2.	(1.2, 3)	(B, B, B)	0.33
3.	(1.3, 4)	(C, C, C)	0.33

Figure D.6. Characteristic Signal Sets for Non-Singularities in VTG-1

The frequency map for all singularities over all single period CSSs is shown in Fig. D.5. This figure does not include elements of the grammar that are non-singularities.

D.2 Frequency Map Calculation, Phase II

Once the singularity frequency map has been calculated and cached, the non-singularity frequencies are calculated. As described in Chapter D.1, the calculation of frequencies for the singularities is exhaustive in nature. As a result, frequency maps for singularities do not consist of any approximations or estimates and are very accurate.

For non-singularities, it is often not possible to maintain this level of precision because it is not possible to exhaustively compute every possible CSS for the non-singularity. This is because the typical grammar can generate an infinite number of distinct strings of non-singularities. As a result, it is necessary to generate a sample set of specific problem instances and to calculate the frequency map from the sample set, giving each sample instance equal weighting.

The detailed process for computing frequency maps for non-singularities is similar to that for singularities. First, a set of sample CSSs is generated. Then, the frequencies of the nonterminals is computed, using cached singularity values, and combined into a domain frequency map.

D.2.1 Non-Singularity CSS Calculation

As with singularities, a set of CSSs is calculated and each is given a distribution factor. In cases where it is possible to exhaustively calculate the set of CSSs, the distribution factor is computed from the grammar as it was for singularities. In cases where a sample set is used, the distribution factor is proportional to the number of elements in the set.

For the grammar VTG-1, the CSSs for non-singularities are shown in Fig. D.6. Note that, for non-singularities, an individual CSS is not associated with a distinct non-singularity but with a derivation tree (i.e., a sequence of grammar rules) that that halts with root singularities. Thus, the non-singularities are generated with a version of the grammar that, effectively, uses the root singularities as terminal symbols.

D.2.2 Non-Singularity Frequency Map Calculation

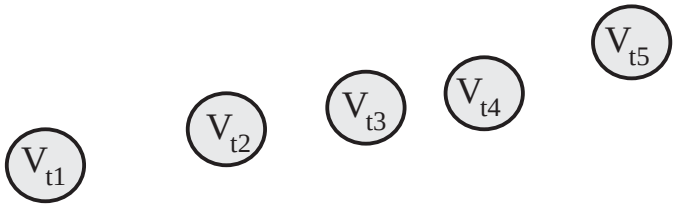
A distinct frequency map is computed for each CSS and the set of frequency maps are then combined into the frequency map for the domain. The computational process is more difficult, however, because, as noted in Chapter 5.1.3, the fact that a non-singularity spans multiple time/locations must be considered. Consider the situation shown in Fig. D.7. In this example, the grammar element t has a RHS consisting of V_t and V_{t+1}, i.e., t $\rightarrow$ V_t V_{t+1}. Given five sequential instances of V, it is possible to combine them in four distinct ways, as shown. If t is treated as a singularity, the frequency, from Def. 5.1.3 would be computed based on Frequency(V) * Frequency(V). Clearly this is not correct. The actual frequency of t should be the frequency of V V combinations, multiplied by the number of combinations of different V_t V_{t+1} combinations. Thus, as shown in the example, there are four combinations of V V; V_1 V_2, V_2 V_3, V_3 V_4, and V_4 V_5. Thus, the actual frequency of t is 4 * (Frequency(V) * Frequency(V)).

As discussed in the thesis, we address this problem by mapping IDP_g to a new grammar, IDP_g', and then computing the frequency of the elements of the new grammar. Specifically, the frequency of each element n of the VTG-1 interpretation grammar is determined by mapping rules containing n as the LHS to a new grammar, VTG-1', and then summing the frequencies of the elements of VTG-1' corresponding to n. The relevant rules in VTG-1' are generated as follows. For each sequential time interval (t_x, t_y) that can be formed using rules with n as the LHS, create a new rule in VTG-1' of the form $n_{x.y} \rightarrow RHS$. In addition, all root singularities are marked with a subscript indicating the period with which they are associated. When combined, elements of the transformed grammar must be consistent in their period subscripts. Then Frequency(n) = $\sum_{x.y}$ Frequency($n_{x.y}$), where $n_{x.y}$ is treated as a singularity.

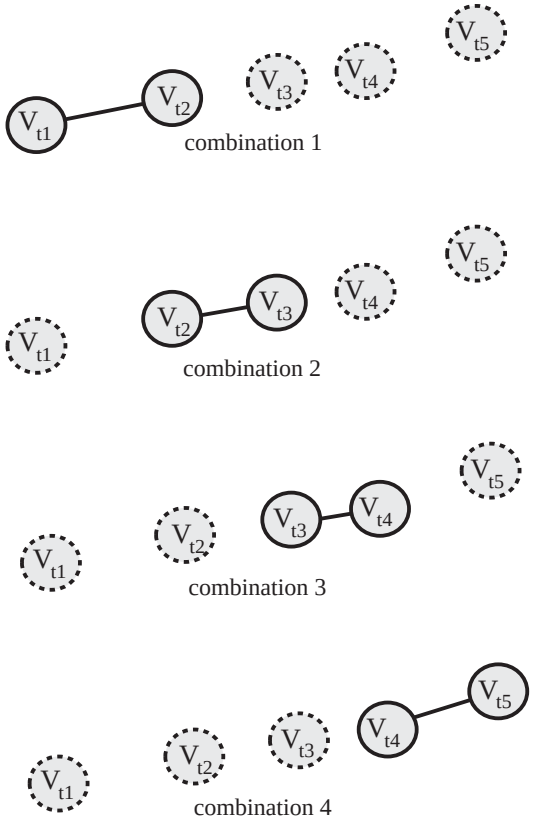
The transformed grammar for VTG-1 is shown in Fig. D.8. Rules P.2.a and P.2.b were added to represent two possible ways of generating a T1 interpretation from a sequence of A's. Rule P.2.a uses the first two A's in a sequence and rule P.2.b uses the second two A's in a sequence. Rules P.3.a and P.3.b serve the same purpose for T2 as do rules P.4.a and P.4.b for T3. In addition, rules P.1.1, P.1.2, and P.1.3 have been modified to indicate that a T can only be generated with a $T1_{1-2}$, $T2_{1-2}$, or $T3_{1-2}$. Thus, the nonterminal symbols $T1_{2-3}$, $T2_{2-3}$, and $T3_{2-3}$ are not connected to the start symbol. Consequently, any effort directed toward their creation or extension is wasted effort. Conceivably, it would be possible for a problem solver to conduct some form of preprocessing to identify nonterminals that weren't connected and to eliminate them from the grammar.

For the CSSs of the non-singularities in VTG-1, the individual frequency maps are shown in Fig. D.9. Each row of the figure contains the non-singularity CSS, the distribution factor for the specific CSS, and the frequency map for the elements of VTG-1'. The individual frequency elements in the map are calculated for each element, n, of the interpretation grammar VTG-1'. For a given production rule, p, for which n is the left-hand-side element, n's frequency is calculated using the same two step process used for singularities. First, the frequency of n from each of the RHSs of p is calculated as the product of the frequencies of the elements of the RHSs, or $\prod_j F(e_j)$, where each e_j is an element of the RHS. Second, the frequencies from each of the RHSs of p are combined by a function that is specific for n. The combination functions are "addition." Thus, the frequency of n from p is the sum of the frequencies from each of the RHSs of p. In the case where n is a terminal symbol, its frequency corresponds to the number of occurrences in the CSS.

For example, for CSS 1, the frequency of $T1_{1-2}$ is computed from Frequency(A given CSS 1) * Frequency(A given CSS 1), or $1.1 * 1.1$ based on rule P.2.a from the grammar



a. initial vehicle level data from 5 time periods



b. all possible partial tracks sequences spanning two consecutive time periods generated by the rule t -> VV

Figure D.7. Calculating the Frequency of Non-Singularities

P.1.1.	T	$\rightarrow T1_{1-2}\ A_3$
P.1.2.	T	$\rightarrow T2_{1-2}\ B_3$
P.1.3.	T	$\rightarrow T3_{1-2}\ C_3$
P.2.a.	$T1_{1-2}$	$\rightarrow A_1\ A_2$
P.2.b.	$T1_{2-3}$	$\rightarrow A_2\ A_3$
P.3.a.	$T2_{1-2}$	$\rightarrow B_1\ B_2$
P.3.b.	$T2_{2-3}$	$\rightarrow B_2\ B_3$
P.4.a.	$T3_{1-2}$	$\rightarrow C_1\ C_2$
P.4.b.	$T3_{2-3}$	$\rightarrow C_2\ C_3$
P.5.	A	$\rightarrow$ G1 G2
P.6.	B	$\rightarrow$ G3 G4
P.7.	C	$\rightarrow$ G5 G6
P.8.1.	G1	$\rightarrow$ S1 S2
P.8.2.	G1	$\rightarrow$ S1 S2 S6
P.9.	G2	$\rightarrow$ S11 S12
P.10.1.	G3	$\rightarrow$ S1 S11
P.10.2.	G3	$\rightarrow$ S1 S11 S12
P.11.	G4	$\rightarrow$ S2 S6
P.12.1.	G5	$\rightarrow$ S2 S11
P.12.2.	G5	$\rightarrow$ S1 S2 S11
P.13.	G6	$\rightarrow$ S6 S12

Figure D.8. Modified Grammar, VTG-1', for Calculating Non-Singularity Frequencies

Non-Singularity CSS	Dist	T	$T1_{1-2}$	$T1_{2-3}$	$T2_{1-2}$	$T2_{2-3}$	$T3_{1-2}$	$T3_{2-3}$
1 (A, A, A)	0.33	1.35	1.21	1.21	0.04	0.04	0.04	0.04
2 (B, B, B)	0.33	1.35	0.04	0.04	1.21	1.21	0.04	0.04
3 (C, C, C)	0.33	1.35	0.04	0.04	0.04	0.04	1.21	1.21

Figure D.9. CSS Frequency Maps for the Transformed Grammar, VTG-1'

VTG-1' in Fig. D.8. The frequency of $T2_{1-2}$ for the same CSS is Frequency(B given CSS 1) * Frequency(B given CSS 1), or $0.2 * 0.2$, based on rule P.3.a from VTG-1'. The frequency of T is the sum of the frequencies from rules P.1.1, P.1.2, and P.1.3. The frequency of T from P.1.1 is Frequency($T1_{1-2}$ given CSS 1) * Frequency(A given CSS 1), or $1.21 * 1.1 = 1.33$. The frequency of T from P.1.2 is Frequency($T2_{1-2}$ given CSS 1) * Frequency(B given CSS 1), or $0.04 * 0.2 = 0.008$. The frequency of T from P.1.3 is Frequency($T3_{1-2}$ given CSS 1) * Frequency(C given CSS 1), or $0.04 * 0.2 = 0.008$. Finally, the frequency of T is $1.33 + 0.008 + 0.008 = 1.35$.

The CSS frequency maps for VTG-1' are combined into the non-singularity frequency map for VTG-1 shown in Fig. D.10. The non-singularity frequency map is formed by summing the corresponding frequencies from VTG-1 for each of the non-singularities. Thus, for CSS 1,

Non-Singularity CSS	Distribution Factor	T	T1	T2	T3
1	0.33	1.35	2.42	0.08	0.08
2	0.33	1.35	0.08	2.42	0.08
3	0.33	1.35	0.08	0.08	2.42
(Totals)	1.00	1.35	0.86	0.86	0.86

Figure D.10. Frequency Map for Non-Singularities in Grammar VTG-1

	T	T1	T2	T3	A	B	C	G1	G2
F_n^B	1.35	0.86	0.86	0.86	1.5	1.5	1.5	3.3	2.1
	G3	G4	G5	G6	S1	S2	S6	S11	S12
F_n^B	3.3	2.1	5.1	1.2	2.1	3.0	2.1	3.0	2.1

Figure D.11. Domain Frequency Map for VTG-1

the frequency of T1 in VTG-1 is the sum of the frequencies of $T1_{1-2}$ and $T1_{2-3}$ in VTG-1'.

Finally, the domain frequency map is created by combining the singularity and non-singularity frequency maps. This is shown in Fig. D.11. For non-singularities, the domain frequencies are computed for each CSS multiplying the values in the non-singularity frequency map by the CSS's distribution factor and then summing over all CSSs. For singularities, the frequencies are computed by multiplying the singularity frequencies from Fig. D.5 by the expected number of periods. In this example, the expected number of periods for VTG-1 is 3. Thus, in Fig. D.11, the entry for A is $3 * 0.5 = 1.5$.

D.3 Frequency Maps and Approximate Processing

In Chapter 13, we present experimental results that are based on a grammar that has been modified to support approximate processing. The processes used to calculate expected costs in these experiments is slightly modified from those described in Chapters D.1 and D.2. The modifications are minor and are described in Chapter 5.1. In this section, we present a detailed example of the processes used to calculate frequency maps for the portions of a grammar associated with approximate processing.

Figure D.12 shows the modifications that were made to the grammar to support approximate processing. Rules LH.1, LH.2, and LH.3 support level hopping, rule C.1 is a clustering operator, and rules AP.1 and AP.2 are rules that process approximations in an abstract projection space. The results of AP.1 are mapped back to the base space via rule M.G.1.

The basic procedures for calculating the frequency map for the grammar extensions that support approximate processing are identical to those for base space operators. The grammar elements are separated into singularities and non-singularities and appropriate CSSs are determined for each. Figure D.13 shows the CSSs and the associated frequency maps for the meta-level singularities. Note that the frequencies in this map are, at most, 1. This observation is related to the only difference between the calculations for base space frequencies

AP.1.	$AT_{c1 \cap c2}$	$\rightarrow AT1_{c1}\ VLC_{c2}$
AP.2.	$AT1_{c1 \cap c2}$	$\rightarrow VLC_{c1}\ VLC_{c2}$
C.1.	$VLC_{A \cup B \cup C}$	$\rightarrow A^{lh} \ldots B^{lh} \ldots C^{lh}$
LH.1.	A^{lh}	$\rightarrow$ S1 S2 S6 S11 S12
LH.2.	B^{lh}	$\rightarrow$ S1 S2 S6 S11 S12
LH.3.	C^{lh}	$\rightarrow$ S1 S2 S6 S11 S12
M.G.1.	$\emptyset$	$\rightarrow$ AT

Figure D.12. Approximations Used to Extend VTG-1

CSS	Dist	A^{lh}	B^{lh}	C^{lh}	$VLC_{A \cup B \cup C}$
1	0.9	1			1
2	0.1	1	1	1	1
3	0.9		1		1
4	0.1	1	1	1	1
5	0.9			1	1
6	0.1	1	1	1	1

Figure D.13. CSS Frequency Maps for Meta-Level Singularities

and meta-level frequencies. In general, most projection operators are clustering operators that inherently aggregate all the possible individual results into a single, abstract result. As a consequence, the frequencies of these states is never greater than 1 in a given period.

Figure D.14 shows the meta-level equivalent of root singularities. Each row in the figure contains the name of the element from IDP_g that corresponds to the CSS and a set of frequencies. Thus, when the domain event that generated the CSS is an A, it will be possible to form an A^{lh} 100 percent of the time. However, as discussed above, there will be at most 1 of these partial results generated. Similarly, when the CSS is generated by an A, it will be possible to form B^{lh} and C^{lh} approximations 10 percent of the time. In all circumstances, a $VLC_{A \cup B \cup C}$ approximation will be formed and the entries in the table reflect this.

It is interesting to compare the entries in this table from those of the corresponding root singularities in the base space shown in Fig. D.4. In Fig. D.4, the frequency entries represent situations where there are multiple instances of each of the root singularities. Since the meta-level singularities never have frequencies greater than 1 in a given period, their frequencies are less than those of base space singularities.

	A^{lh}	B^{lh}	C^{lh}	$VLC_{A \cup B \cup C}$
A	1.0	0.1	0.1	1.0
B	0.1	1.0	0.1	1.0
C	0.1	0.1	1.0	1.0

Figure D.14. Frequency Map for Approximations as Related to Root Singularities in Grammar VTG-1

	$VLC_{A \cup B \cup C}$	A^{lh}	B^{lh}	C^{lh}
F_n^B	1.0	0.4	0.4	0.4

Figure D.15. Approximation Singularity Frequency Map for VTG-1

Figure D.15 shows the frequency map for meta-level states in VTG-1 combined over all CSSs. The entry for $VLC_{A \cup B \cup C}$ is 1 because in each period, all the level-hopping results are aggregated into a single abstract cluster.

The process for calculating the frequencies of meta-level non-singularities starts with the same grammar transformation used in the base space calculations. The transformed elements of the grammar are shown in Fig. D.16. The only rules that are transformed correspond to the rules that are used to construct an abstract solution in a projection space. The original rule AP.2 is represented by two new rules, AP.2.a and AP.2.b, that correspond to the two possible combinations of sequences of VLC partial results.

All the elements of the transformed grammar are treated as singularities and the frequency map shown in Fig. D.17 is computed. There is no restriction on meta-level states formed with AP operators that precludes their frequencies from being greater than 1. Instead, in this example, the AP operators are being applied to clustered data and the frequency of the clusters is always 1. Consequently, the combining function which multiplies the frequency of the components simply multiplies $1 * 1$ and the resulting frequency is always 1.

The frequency map of the transformed grammar is used to construct the frequency map for meta-level non-singularities shown in Fig. D.18. The process used is identical to that used for the base space processing.

Finally, the complete meta-level frequency map is computed by combining the frequency maps for meta-level singularities and non-singularities. This is shown in Figure D.19.

AP.1.	$AT_{c1 \cap c2}$	$\rightarrow AT1_{c1,1-2}\ VLC_{c2,3}$
AP.2.a	$AT1_{c1 \cap c2,1-2}$	$\rightarrow VLC_{c1,1}\ VLC_{c2,2}$
AP.2.b	$AT1_{c1 \cap c2,2-3}$	$\rightarrow VLC_{c1,2}\ VLC_{c2,3}$

Figure D.16. Transformed Approximations For Frequency Computation

Non-Singularity CSS	Dist	$AT_{c1 \cap c2}$	$AT1_{c1 \cap c2,1-2}$	$AT1_{c1 \cap c2,2-3}$
1 (A, A, A)	0.33	1.0	1.0	1.0
2 (B, B, B)	0.33	1.0	1.0	1.0
3 (C, C, C)	0.33	1.0	1.0	1.0

Figure D.17. CSS Frequency Maps for the Meta-Level Non-Singularities in the Transformed Grammar, VTG-1'

Non-Singularity CSS	Distribution Factor	$AT_{c1 \cap c2}$	$AT1_{c1 \cap c2}$
1	0.33	1.0	2.0
2	0.33	1.0	2.0
3	0.33	1.0	2.0
(Totals)	1.00	1.0	2.0

Figure D.18. Frequency Map for Meta-Level Non-Singularities in Grammar VTG-1

D.4 Approximate Processing and Base Space Frequency Maps

In Chapter 13, we describe a set of experiments that were conducted to validate the application of the IDP/*UPC* framework in the analysis of approximate processing domains. In these experiments, the frequency maps for meta-levels were calculated as described in Chapter D.3. Chapter D.3 did not discuss the effects that approximate processing has on the base space or on the frequencies of base space elements. These issues are discussed in detail in this section.

Chapter 10.3.4 describes a variety of methods for mapping the results of processing in a projection space back to the base space. In the experiments in Chapter 13, the problem solver used a grammar transformation to map the results of problem solving back to the base space. After using approximate processing to determine a set of potential solutions, the problem solver transforms the interpretation grammar by removing all rules that are not on paths to elements in the set of potential solutions. We will now illustrate the processes used to transform the grammar and to compute the resulting frequency map.

The grammar used in the experiments is shown again in Fig. D.20. Now, consider the problem instance shown in Fig. D.21. The input data is shown with subscripts indicating the time period in which the signal was detected.

	$AT_{c1 \cap c2}$	$AT1_{c1 \cap c2}$	$VLC_{A \cup B \cup C}$	A^{lh}	B^{lh}	C^{lh}
F_n^B	1.0	2.0	3.0	1.2	1.2	1.2

Figure D.19. Meta-Level Frequency Map for VTG-1

P.1.1.	T	→ T1 A	p=0.33
P.1.2.	T	→ T2 B	p=0.33
P.1.3.	T	→ T3 C	p=0.33
P.2.	T1	→ A A	p=1.0
P.3.	T2	→ B B	p=1.0
P.4.	T3	→ C C	p=1.0
P.5.	A	→ G1 G2	p=1.0
P.6.	B	→ G3 G4	p=1.0
P.7.	C	→ G5 G6	p=1.0
P.8.1.	G1	→ S1 S2	p=0.9
P.8.2.	G1	→ S1 S2 S6	p=0.1
P.9.	G2	→ S11 S12	p=1.0
P.10.1.	G3	→ S1 S11	p=0.9
P.10.2.	G3	→ S1 S11 S12	p=0.1
P.11.	G4	→ S2 S6	p=1.0
P.12.1.	G5	→ S2 S11	p=0.9
P.12.2.	G5	→ S1 S2 S11	p=0.1
P.13.	G6	→ S6 S12	p=1.0

Figure D.20. Full Grammar VTG-1 for Tracking Vehicles Through Multiple Time-Locations

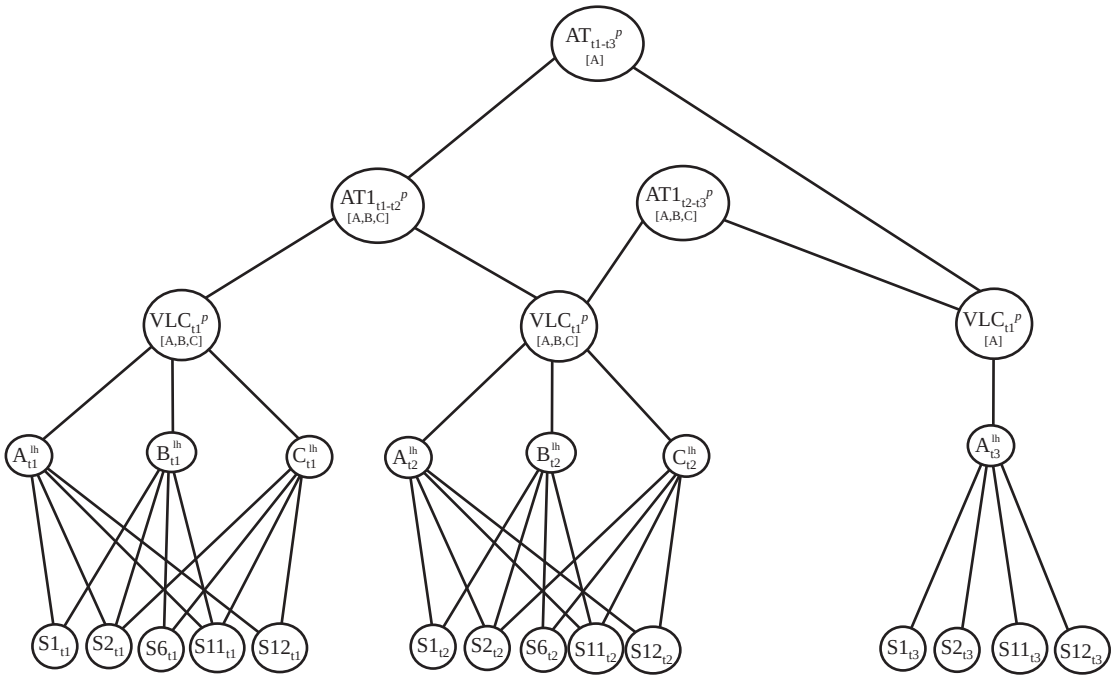


Figure D.21. Example Problem Instance

P.1.1.	T	→ T1 A	p=1.0
P.2.	T1	→ A A	p=1.0
P.5.	A	→ G1 G2	p=1.0
P.8.1.	G1	→ S1 S2	p=0.9
P.8.2.	G1	→ S1 S2 S6	p=0.1
P.9.	G2	→ S11 S12	p=1.0

Figure D.22. VTG-1 Transformed By Mapping Operator

CSS Num	Singularity	Rules	Characteristic Signal Set	Distribution Factor
1.	A	(5, 8.1, 9)	(S1, S2, S11, S12)	0.9
2.	A	(5, 8.2, 9)	(S1, S2, S6, S11, S12)	0.1

Figure D.23. Characteristic Signal Sets for Transformed VTG-1

The figure shows the approximate processing that is conducted on the input. Level hopping approximations are clustered at an abstract level and combined into an approximate track. The approximate track result indicates that the characteristics of the potential solution set are restricted to tracks of vehicles of type A. This result is mapped back to the base space by transforming the base space grammar to that shown in Fig. D.22. The grammar is transformed by eliminating all paths that lead to results are not in the potential solution set. Thus, all paths to other than A are eliminated.

Now, the computation of the frequency map is identical to that described in the preceding sections as illustrated in the following figures.

Figure D.23 shows the CSSs for the transformed grammar. Note that all CSSs other than those that can be generated from an A are deleted.

Sing.	CSS	Dist	A	B	C	G1	G2	G3	G4	G5	G6	S1	S2	S6	S11	S12
A	1	0.9	1			1	1					1	1		1	1
A	2	0.1	2			2	1					1	1	1	1	1

Figure D.24. CSS Frequency Map for Singularities in the Transformed Grammar

CSS Num	CSS Generation Rules	Characteristic Signal Set	Distribution Factor
1.	(1.1, 2)	(A, A, A)	1.0

Figure D.25. Characteristic Signal Sets for Non-Singularities in Transformed VTG-1

Figure D.24 shows the CSS frequency maps for the elements of the transformed grammar. Although many of the elements shown have actually been deleted in the transformed grammar, they are left in the figure to contract the transformed frequency map with the original frequency map in Fig. D.3. For example, in the second row, the entries for B and C are both 0. In the original frequency map, they were both 2. However, since neither B or C is on a path to an "A" track, these partial results are not generated by the transformed grammar.

In the transformed grammar, there is only a single non-singularity CSS and it is shown in Fig. D.25.

The transformed grammar is still modified to compute the frequencies of non-singularities as shown in Fig. D.26.

The frequencies of the modified version of the transformed grammar are shown in Fig. D.27.

The modified CSS frequencies are converted to the frequencies for the non-singularities as shown in Fig. D.28.

Finally, the domain frequency map for the transformed grammar is created by combining the singularity and non-singularity frequency maps. This is shown in Fig. D.29.

P.1.1.	T	$\rightarrow T1_{1-2}\ A_3$
P.2.a.	$T1_{1-2}$	$\rightarrow A_1\ A_2$
P.2.b.	$T1_{2-3}$	$\rightarrow A_2\ A_3$
P.5.	A	$\rightarrow$ G1 G2
P.8.1.	G1	$\rightarrow$ S1 S2
P.8.2.	G1	$\rightarrow$ S1 S2 S6
P.9.	G2	$\rightarrow$ S11 S12

Figure D.26. Modified Version of Transformed Grammar for Calculating Non-Singularity Frequencies

Non-Singularity CSS	Dist	T	$T1_{1-2}$	$T1_{2-3}$
1 (A, A, A)	0.33	1.33	1.21	1.21

Figure D.27. CSS Frequency Maps for the Transformed Grammar, VTG-1'

Non-Singularity CSS	Distribution Factor	T	T1
1	1.0	1.33	2.42

Figure D.28. Frequency Map for Non-Singularities in the Transformed VTG-1

D.5 Discussion

It should be clear from the preceding examples that the concepts of singularities and non-singularities are important components of the IDP/*UPC* framework. The use of singularities makes it possible to accurately compute frequency maps of even very large, complex domains. More specifically, the use of singularities makes it possible to generate frequency maps based on exhaustive CSS enumeration in many instances where it would not otherwise be possible to do so. If one tried to compute these frequencies maps without the use of singularities, it would be necessary to estimate frequency values based on sample sets and this would inherently skew the results with sampling errors.

A very important contribution associated with the use of singularities is that of computational efficiency. By computing singularity values and then using cached information, it is possible to quickly calculate analyses for large, complex grammars.

In the vehicle tracking domain, the definition of singularities is naturally associated with the time periods that are characteristic of the problem. However, it is possible to define singularities for virtually any interpretation domain. In essence, a singularity is a component of a problem domain that is repeated multiple times within a single problem instance. For example, in a natural language (NLP) or image processing (IP) domain, it is possible to define many kinds of singularities. Verb Phrases, noun phrases, sentences, and paragraphs are all example of singularities that can be found in NLP domains. Images of people, faces, cars, etc., are examples of singularities that are found in image processing domains.

An important consideration in designing a domain grammar is that the elements of a singularity should be as heterogeneous as possible. For example, if a verb phrase singularity were defined for an NLP domain, it might be necessary to define two different kinds of verb

	T	T1	A	G1	G2	S1	S2	S6	S11	S12
F_n^B	1.33	2.42	3.3	3.3	3.0	3.0	3.0	0.1	3.0	3.0

Figure D.29. Domain Frequency Map for VTG-1

phrases, active and passive. This is sometimes necessary to get meaningful differentiation in an analysis. For example, in a vehicle tracking domain, it is not necessarily useful to lump all vehicle types into a single grammar element. This makes it impossible to differentiate the behavior of distinct kinds of tracks in the analysis.

REFERENCES

[Berliner, 1979] Hans Berliner. The B* tree search algorithm: A best-first proof procedure. *Artificial Intelligence*, 12:23–40, 1979.

[Bonissone and Decker, 1986] Piero P. Bonissone and Keith S. Decker. Selecting uncertainty calculi and granularity: An experiment in trading-off precision and complexity. In L. N. Kanal and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence*. North Holland, 1986.

[Bonissone *et al.*, 1987] Piero P. Bonissone, Steven S. Gans, and Keith S. Decker. RUM: A layered architecture for reasoning with uncertainty. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, August 1987.

[Campbell *et al.*, 1991] I. C. Campbell, K. J. Luczynski, and I. Hood. Putting knowledge-based concepts to work for mechanical design. In Reid Smith and Carlisle Scott, editors, *Innovative Applications of Artificial Intelligence 3: Proceedings of the IAAI-91 Conference*, pages 157–176. AAAI Press/The MIT Press, 1991. ISBN 0-262-68068-8.

[Carver and Lesser, 1991] Norman Carver and Victor Lesser. The Evolution of Blackboard Control. *Expert Systems with Applications*, 7(1), 1991. Special issue on The Blackboard Paradigm and Its Applications.

[Carver and Lesser, 1993] Norman Carver and Victor R. Lesser. Planning for the control of an interpretation system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6), 1993. Special Issue on Scheduling, Planning, and Control.

[Corkill and Lesser, 1981] Daniel D. Corkill and Victor R. Lesser. A goal-directed Hearsay-II architecture: Unifying data-directed and goal-directed control. Technical Report 81-15, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, June 1981.

[Corkill *et al.*, 1982] Daniel D. Corkill, Victor R. Lesser, and Eva Hudlická. Unifying data-directed and goal-directed control: An example and experiments. In *Proceedings of the National Conference on Artificial Intelligence*, pages 143–147, Pittsburgh, Pennsylvania, August 1982.

[Corkill, 1983] Daniel David Corkill. *A Framework for Organizational Self-Design in Distributed Problem Solving Networks*. PhD thesis, University of Massachusetts, February 1983. (Also published as Technical Report 82-33, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, December 1982.).

[Davis, 1980] Randall Davis. Meta-rules: Reasoning about control. *Artificial Intelligence*, 15:179–222, 1980.

[Decker *et al.*, 1989] Keith Decker, Marty Humphrey, and Victor Lesser. Experimenting with control in the DVMT. Technical Report 89-00, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, March 1989.

[Decker *et al.*, 1990] Keith S. Decker, Victor R. Lesser, and Robert C. Whitehair. Extending a blackboard architecture for approximate processing. *The Journal of Real-Time Systems*, 2(1/2):47–79, 1990. Also COINS TR-89-115.

[Durfee and Lesser, 1986] Edmund H. Durfee and Victor R. Lesser. Incremental planning to control a blackboard-based problem solver. In *Proceedings of the National Conference on Artificial Intelligence*, pages 58–64, Philadelphia, Pennsylvania, August 1986.

[Durfee, 1987] Edmund H. Durfee. *A Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network*. PhD thesis, University of Massachusetts, September 1987. (Also published as Technical Report 87-84, Department of Computer and Information Science, University of Massachusetts, Amherst, MA, September,1987.).

[Erman *et al.*, 1980] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.

[Fox, 1983] Mark S. Fox. *Constraint-directed Search: A Case Study of Job-Shop Scheduling*. PhD thesis, Carnegie-Mellon University, 1983. (Also published as Technical Report CMU-CS-83-161, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213.).

[Fu, 1982] King Sun Fu. *Syntactic Pattern Recognition and Applications*. PH, 1982.

[Garvey and Lesser, 1993] Alan Garvey and Victor Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1491–1502, 1993.

[Gazdar *et al.*, 1982] Gerald Gazdar, Geoffrey K. Pullum, and Ivan A. Sag. Auxiliaries and related phenomena in a restrictive theory of grammar. *Language*, 58(3):591–638, 1982.

[Genesereth and Smith, 1982] Michael R. Genesereth and David E. Smith. Meta-level architecture. Technical report, Computer Science Department, Stanford University, Stanford, California 94305, December 1982.

[Genesereth, 1983] Michael R. Genesereth. An overview of meta-level architecture. In *Proceedings of the National Conference on Artificial Intelligence*, pages 119–124, Washington, D.C., August 1983.

[Hayes-Roth and Hayes-Roth, 1979] Barbara Hayes-Roth and Frederick Hayes-Roth. A cognitive model of planning. *Cognitive Science*, 3(4):275–310, October–December 1979.

[Hayes-Roth and Lesser, 1977] Frederick Hayes-Roth and Victor R. Lesser. Focus of attention in the Hearsay-II system. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 27–35, Tiblisi, Georgia, USSR, August 1977.

[Hayes-Roth, 1985] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, July 1985.

[Hudlická and Lesser, 1984] Eva Hudlická and Victor R. Lesser. Meta-level control through fault detection and diagnosis. In *Proceedings of the National Conference on Artificial Intelligence*, pages 153–161, Austin, Texas, August 1984.

[III, 1990] Norman F. Carver III. *Sophisticated Control for Interpretation: Planning to Resolve Sources of Uncertainty*. PhD thesis, University of Massachusetts, September 1990.

[Johnson and Hayes-Roth, 1987] M. Vaughn Johnson and Barbara Hayes-Roth. Integrating diverse reasoning methods in the BB1 blackboard control architecture. In *Proceedings of the National Conference on Artificial Intelligence*, pages 30–35, Seattle, Washington, July 1987.

[Knoblock, 1991a] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving*. PhD thesis, Carnegie Mellon University, 1991. (Also published as Technical Report CMU-CS-91-120, School of Computer Science, Carnegie Mellon University.).

[Knoblock, 1991b] Craig A. Knoblock. Search reduction in hierarchical problem solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 686–691, San Diego, California, July 1991.

[Knoblock, 1994] Craig A. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.

[Knuth, 1968] D. Knuth. Semantics of context-free grammars. *Mathematical Systems Theory*, 2(2):127–145, 1968.

[Kumar and Kanal, 1988] Vipin Kumar and Laveen N. Kanal. The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In L. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, Symbolic computation, chapter 1, pages 1–27. Springer-Verlag, 1988.

[Lesser and Corkill, 1983] Victor Lesser and Daniel Corkill. The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3):15–33, Fall 1983. (Also to appear in *Blackboard Systems*, Robert S. Engelmore and Anthony Morgan, editors, Addison-Wesley, in press, 1988 and in *Readings from AI Magazine 1980–1985*, in press, 1988).

[Lesser and Pavlin, 1988] Victor R. Lesser and Jasmina Pavlin. Performing approximate processing to address real-time constraints. COINS Technical Report 87-126, University of Massachusetts, 1988.

[Lesser *et al.*, 1987] Victor R. Lesser, Daniel D. Corkill, and Edmund H. Durfee. An update on the Distributed Vehicle Monitoring Testbed. Technical Report 87-111, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, December 1987.

[Lesser *et al.*, 1988a] Victor Lesser, Jasmina Pavlin, and Edmund Durfee. Approximate processing in real-time problem solving. *AI Magazine*, 9(1):49–61, Spring 1988.

[Lesser *et al.*, 1988b] Victor R. Lesser, Jasmina Pavlin, and Edmund Durfee. Approximate processing in real-time problem solving. *AI Magazine*, 9(1):49–61, Spring 1988.

[Lesser *et al.*, 1989a] V. R. Lesser, D. D. Corkill, R. C. Whitehair, and J. A. Hernandez. Focus of control through goal relationships. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, August 1989.

[Lesser *et al.*, 1989b] V. R. Lesser, R. C. Whitehair, D. D. Corkill, and J. A. Hernandez. Goal relationships and their use in a blackboard architecture. In V. Jagannathan, Rajendra Dodhiawala, and Lawrence Baum, editors, *Blackboard Architectures and Applications*, pages 9–26. Academic Press, Inc., 1989.

[Lesser *et al.*, 1993] Victor Lesser, Hamid Nawab, Izaskun Gallastegi, and Frank Klassner. IPUS: An architecture for integrated signal processing and signal interpretation in complex environments. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, DC, July 1993.

[Lowrance and Garvey, 1982] John D. Lowrance and Thomas D. Garvey. Evidential reasoning: A developing concept. *IEE 1982 Proceedings of the International Conference on Cybernetics and Society*, pages 6–9, 1982.

[Minsky, 1963] Marvin Minsky. Steps towards artificial intelligence. In E. A. Fiegenbaum and J. Feldman, editors, *Computers and Thought*, pages 406–450. McGraw-Hill, 1963.

[Mullins and Rinderle, 1991a] Scott Mullins and James Rinderle. Grammatical approaches to engineering design, part I: An introduction and commentary. *Research in Engineering Design*, 2:121–135, 1991.

[Mullins and Rinderle, 1991b] Scott Mullins and James Rinderle. Grammatical approaches to engineering design, part II: Melding configuration adn parametric design using atrribute grammars. *Research in Engineering Design*, 2:137–146, 1991.

[Newell *et al.*, 1962] A. Newell, J. C. Shaw, and H. a. Simon. The process of creative thinking. In *Contemporary Approaches to Creative Thinking*, pages 63–119. Atherton Press, New York, 1962.

[Newell *et al.*, 1963] A. Newell, J. C. Shaw, and H. a. Simon. Empirical explorations with the logic theory machine: A case history of heuristics. In E. A. Fiegenbaum and J. Feldman, editors, *Computers and Thought*, pages 109–133. McGraw-Hill, 1963.

[Papadimitriou and Steiglitz, 1982] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.

[Pearl, 1984] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, first edition, 1984.

[Samuel, 1963] A. L. Samuel. Some studies in machine learning using the game of checkers. In E. A. Fiegenbaum and J. Feldman, editors, *Computers and Thought*, pages 71–105. McGraw-Hill, 1963.

[Simon, 1969] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, 1969.

[Stefik, 1981] Mark Stefik. Planning and meta-planning (MOLGEN: Part 2). *Artificial Intelligence*, 16:141–170, 1981.

[Stockman, 1979] G. C. Stockman. A minimax algorithm better than Alpha-Beta? *Artificial Intelligence*, 12:179–196, 1979.

[V.R.Lesser *et al.*, 1975] V.R.Lesser, R.D.Fennell, L.D.Erman, and D.R.Reddy. Organization of the hearsay-ii speech understanding system. In *IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-23*, pages 11–23, January 1975.

[Whitehair and Lesser, 1993] Robert C. Whitehair and Victor R. Lesser. A Framework for the Analysis of Sophisticated Control in Interpretation Systems. Technical Report 93–53, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, 1993.

[Wilensky, 1981] Robert Wilensky. Meta-planning: Representing and using knowledge about planning in problem solving and natural language understanding. *Cognitive Science*, 5(3):197–233, July–September 1981.