IDP: A Formalism for Modeling and Analyzing Complex Interpretation Problem Domains *

Robert C. Whitehair and Victor R. Lesser
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
413-545-3444(voice)
413-545-1249(fax)
Email: whitehair@cs.umass.edu

January 5, 1995

Abstract

This paper presents the *IDP formalism* for representing and analyzing complex interpretation problem domains. The formalism is being used to analyze the relationship between the performance of search-based interpretation problem solving systems and the inherent properties, or *structure*, of problem domains in which they are applied. Models built using this formalism can be used for describing, predicting and explaining the behavior of *interpretation* systems and for generalizing a specific problem solving architecture to other domains. Examples demonstrate how domain phenomena such as noise and missing data are represented, how non-local relationships between subproblems can be modeled, and how quantitative properties of search spaces can be calculated.

Keywords: control, search, mathematical foundations

* This material is based upon work supported by the National Science Foundation under Grant No. IRI-9321324 and the Office of Naval Research contract N00014-92-J-1450. The content does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred. This paper has not already been accepted by and is not currently under review for a journal or another conference. Nor will it be submitted for such during IJCAI's review preiod.

1 Introduction

In this paper, we present formal methods for characterizing the structure of *interpretation problem* domains[1] and their associated problem solving architectures. Our primary assumptions are that the performance of an interpretation problem solver applied in a specific domain is a function of the structure of the problem domain and that effective problem solving techniques can be derived from a formal specification of a problem domain. We define interpretation problems as a search process where, given an input string X, a problem solver attempts to find the most *credible* (or "best") explanation for X from the set I of all possible interpretations (i.e., an interpretation problem is a discrete optimization problem). Interpretation is a form of *constructive problem solving* based on *abductive inferencing*. Interpretation problems are similar to *classification* problems and to a more distant form of problem solving, *parsing*. 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 [2].

For many domains, a distinct 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 to us 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 interpretation trees i in I. 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. Each syntactic rule of the grammar is associated with a corresponding semantic process which determines the actual "meaning" of an interpretation. This representation is similar to that used in the *Composite Decision Process (CDP)* model of Kanal and Kumar [6] and the semantic grammars used by Fu [4].

Based on this definition of interpretation, we define the *Interpretation Decision Problem (IDP)* formalism. The formalism comprises two components, one for modeling the domain phenomena that generate a specific problem instance and one for modeling the structure of an interpretation problem solver. The generational component of the IDP formalism will be referred to as IDP_G and the interpretation component will be referred to as IDP_I . The primary focus of this paper will be on IDP_G and the details of IDP_I are left to [7]. The IDP_G formalism models interpretation *domain theories*[2] in terms of four feature structures: *component* (or *syntax*),

[1] 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 [3].

[2] A domain theory is the computational theory that is the basis for a problem solver's functionality.

utility (or *credibility*)[3], *probability* (or *distribution*), and *cost*. The different feature structures are represented in terms of a *domain grammar* and functions associated with production rules of the grammar. The formal definition of an IDP grammar follows.

Definition 1.1 *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*[4] symbols that correspond to final states in the associated interpretation search space, S is the start symbol for the grammar, and P is the set of context-free production rules.

The component structure is modeled directly by the rules, P, of G. The component subproblems of A are defined by the right-hand-side (RHS) of a rule of the grammar, $p : A \rightarrow$(component subproblems). The full structure is specified recursively and subproblems specified by the nonterminals and terminals of a grammar correspond to states in the associated search space.

The credibility structure is defined recursively in terms of credibility functions associated with G's production rules. Credibility functions include consideration of the credibilities of the component elements and semantics. Given p. A $\rightarrow$ CD, the credibility of "A," f_{A}, is a function of the credibilities of "C" and "D" and the *semantic function*, Γ, that measures the degree to which "C" and "D" are semantically consistent. 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 by syntactic functions. These attributes are represented using the *feature list convention* described by Gazdar, et al. [5]. For example, a semantic process in an acoustic vehicle tracking system might combine two partial vehicle tracks into a longer track using a complex process that verifies the consistency of the frequency characteristics of the partial tracks and the properties of the resulting track such as acceleration and velocity constraints based on time/location attributes of the tracks. Credibility functions are represented $f_{\mathrm{A}}(f_{\mathrm{C}}, f_{\mathrm{D}}, \Gamma_p(\mathrm{C}, \mathrm{D}))$, where the subscript of Γ, p, is the number of the corresponding production rule from the grammar. 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.

Similarly, computational cost structures are recursive functions of component elements and the cost of applying semantic operators. For example, given p. A $\rightarrow$ CD, the cost of generating "A," g_{A}, is a function of the cost of generating "C" and "D" and the cost of applying the semantic function $\Gamma_p(\mathrm{C}, \mathrm{D})$. This is represented $g_{\mathrm{A}}(g_{\mathrm{C}}, g_{\mathrm{D}}, C(\Gamma_p(\mathrm{C}, \mathrm{D})))$, where $C(\Gamma_p(i, j, ...))$ is the cost of applying the semantic function Γ_P.

[3] In [7], 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.

[4] In the IDP formalism, a potential solution is referred to as an SNT. The uses and advantages of the SNT representation are discussed in [7].

The probability structure is defined in terms of the function, ψ. For each $p \in P$, $\psi(p)$ represents the distribution of RHSs associated with p. This can be used both to generate problem instances, in which case ψ models the likelihood of certain events happening in the environment, and to specify the actions available to a problem solver, in which case ψ models the problem solver's perception of and assumptions about the domain. For production p which decomposes to RHS_1, $\text{RHS}_2 \ldots \text{RHS}_m$, $\psi(p)$ models the frequency distribution with which these rules are used to generate problem instances. The frequency with which the rules are used in interpretation is a function of this distribution. Note that $\sum_n \psi(p.n) = 1$. ψ supports models of real-world phenomena such as uncertainty caused by noise, missing data, distortion and masking, as will be discussed in Section 2.

To our knowledge, this formalism is unique and it offers significant contributions to the study of interpretation problems. The most important is that the IDP formalism supports the analysis of the assumptions (implicit and explicit) that a problem solver makes about a domain. For example, using the IDP formalism, it is possible to analyze a problem solver's a priori expectations about a domain. By analyzing the statistical properties of the interpretation trees of a domain grammar, it is possible to determine characteristics that can be used during problem solving, such as a priori expected distributions of domain events that can be used in model-based processing. Furthermore, this analysis can also identify patterns and structures that can be used to construct effective problem solving strategies [7]. In addition, the IDP formalism supports comparative experiments and analyses involving different problem solvers applied in the same domain and the same problem solver applied to different domains. Consequently, this work represents an initial step needed to formalize complex search processes, such as those associated with blackboard systems [1] that are used in sophisticated interpretation systems.

The next section demonstrates how the IDP_G formalism can represent domain phenomena such as noise and missing data. Section 3 describes quantitative results that can be derived using the formalism and presents examples of how these can be used to define general domain structures.

2 Representing Problem Domains

The quality and effectiveness of analysis tools derived from a formal model of a problem domain are dependent on the degree to which the formalism adequately represents the phenomena, and their causes, characteristic of a domain. For interpretation domains, a modeling formalism must deal effectively with interacting phenomena, noise, missing data, distortion, and masking [3]. In this section, we will discuss how the IDP_G formalism addresses these issues.

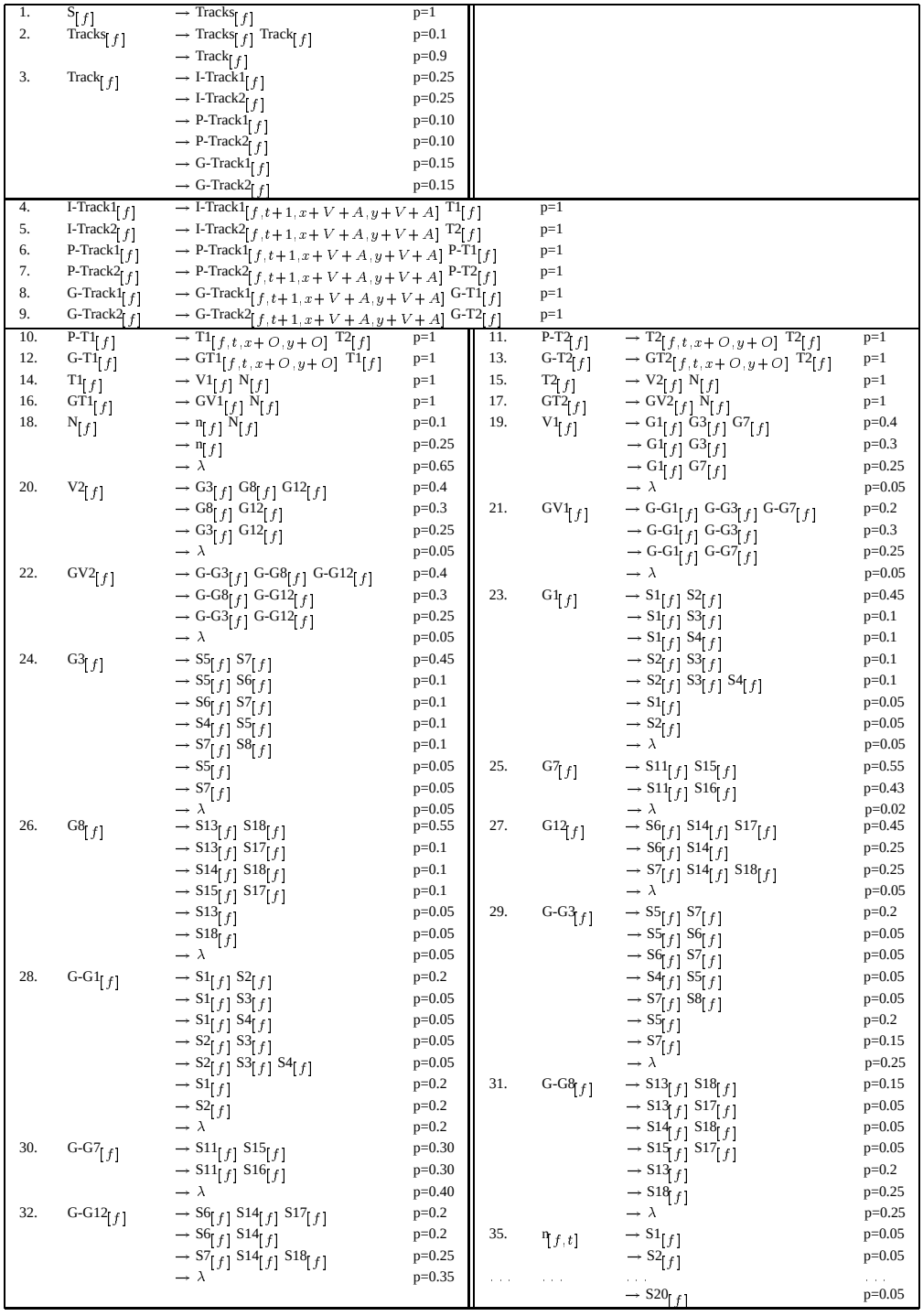
The interpretation domain which we will study in detail is a vehicle tracking problem based on acoustic sensor data as defined in our previous work on RESUN [2]. The problem solver's input consists of preprocessed sensor data gathered from a single contiguous region. The problem solver then processes the data in an attempt to identify the type of vehicle that generated the signals and the path it traversed

through the region. For the vehicle tracking domain, a modeling formalism must also represent multiple events (i.e., multiple vehicle tracks) and patterned events (vehicles moving in a coordinated manner).

In order to model the genertion of a problem instance, the IDP$_G$ simulator begins by creating a feature list for the start symbol, S. This list includes credibility, 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. 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 vehicle tracking problem 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*, Vn, where n is a vehicle identifier. (e.g., V1 is a vehicle 1 location at time t.) 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.

A grammar that will generate problem instances for this domain is shown in Fig. 1. 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. In this example, "f" represents the characteristics time, x and y coordinates, velocity (v), acceleration (a), offset, and energy. Velocity is used to generate the x and y coordinates of the next location in the track. Acceleration 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 represents a displacement from a "base" position. This is used to generate pattern tracks of multiple vehicles and ghost tracks [7]. Figure 1 explicitly shows how feature lists are modified for the different grammar rules. For example, rule 10 indicates that the generation of two track locations from a pattern track location involves adding the offset to the x and y locations of track location T1. Similarly, the generation of a T1 track location from an I-Track1 in rule 4 also involves generating a "new" I-Track1 for the next time quantum (t+1) with new x and y positions (x+V+A, y+V+A). Many rules include a RHS of λ. This indicates that there is a possibilty that the production



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.	$Track_{[f]}$	$\rightarrow$ $I\text{-}Track1_{[f]}$	p=0.25
		$\rightarrow$ $I\text{-}Track2_{[f]}$	p=0.25
		$\rightarrow$ $P\text{-}Track1_{[f]}$	p=0.10
		$\rightarrow$ $P\text{-}Track2_{[f]}$	p=0.10
		$\rightarrow$ $G\text{-}Track1_{[f]}$	p=0.15
		$\rightarrow$ $G\text{-}Track2_{[f]}$	p=0.15
4.	$I\text{-}Track1_{[f]}$	$\rightarrow$ $I\text{-}Track1_{[f,t+1,x+V+A,y+V+A]}$ $T1_{[f]}$	p=1
5.	$I\text{-}Track2_{[f]}$	$\rightarrow$ $I\text{-}Track2_{[f,t+1,x+V+A,y+V+A]}$ $T2_{[f]}$	p=1
6.	$P\text{-}Track1_{[f]}$	$\rightarrow$ $P\text{-}Track1_{[f,t+1,x+V+A,y+V+A]}$ $P\text{-}T1_{[f]}$	p=1
7.	$P\text{-}Track2_{[f]}$	$\rightarrow$ $P\text{-}Track2_{[f,t+1,x+V+A,y+V+A]}$ $P\text{-}T2_{[f]}$	p=1
8.	$G\text{-}Track1_{[f]}$	$\rightarrow$ $G\text{-}Track1_{[f,t+1,x+V+A,y+V+A]}$ $G\text{-}T1_{[f]}$	p=1
9.	$G\text{-}Track2_{[f]}$	$\rightarrow$ $G\text{-}Track2_{[f,t+1,x+V+A,y+V+A]}$ $G\text{-}T2_{[f]}$	p=1
10.	$P\text{-}T1_{[f]}$	$\rightarrow$ $T1_{[f,t,x+O,y+O]}$ $T2_{[f]}$	p=1
11.	$P\text{-}T2_{[f]}$	$\rightarrow$ $T2_{[f,t,x+O,y+O]}$ $T2_{[f]}$	p=1
12.	$G\text{-}T1_{[f]}$	$\rightarrow$ $GT1_{[f,t,x+O,y+O]}$ $T1_{[f]}$	p=1
13.	$G\text{-}T2_{[f]}$	$\rightarrow$ $GT2_{[f,t,x+O,y+O]}$ $T2_{[f]}$	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
		$\rightarrow$ $n_{[f]}$	p=0.25
		$\rightarrow$ λ	p=0.65
19.	$V1_{[f]}$	$\rightarrow$ $G1_{[f]}$ $G3_{[f]}$ $G7_{[f]}$	p=0.4
		$\rightarrow$ $G1_{[f]}$ $G3_{[f]}$	p=0.3
		$\rightarrow$ $G1_{[f]}$ $G7_{[f]}$	p=0.25
		$\rightarrow$ λ	p=0.05
20.	$V2_{[f]}$	$\rightarrow$ $G3_{[f]}$ $G8_{[f]}$ $G12_{[f]}$	p=0.4
		$\rightarrow$ $G8_{[f]}$ $G12_{[f]}$	p=0.3
		$\rightarrow$ $G3_{[f]}$ $G12_{[f]}$	p=0.25
		$\rightarrow$ λ	p=0.05
21.	$GV1_{[f]}$	$\rightarrow$ $G\text{-}G1_{[f]}$ $G\text{-}G3_{[f]}$ $G\text{-}G7_{[f]}$	p=0.2
		$\rightarrow$ $G\text{-}G1_{[f]}$ $G\text{-}G3_{[f]}$	p=0.3
		$\rightarrow$ $G\text{-}G1_{[f]}$ $G\text{-}G7_{[f]}$	p=0.25
		$\rightarrow$ λ	p=0.05
22.	$GV2_{[f]}$	$\rightarrow$ $G\text{-}G3_{[f]}$ $G\text{-}G8_{[f]}$ $G\text{-}G12_{[f]}$	p=0.4
		$\rightarrow$ $G\text{-}G8_{[f]}$ $G\text{-}G12_{[f]}$	p=0.3
		$\rightarrow$ $G\text{-}G3_{[f]}$ $G\text{-}G12_{[f]}$	p=0.25
		$\rightarrow$ λ	p=0.05
23.	$G1_{[f]}$	$\rightarrow$ $S1_{[f]}$ $S2_{[f]}$	p=0.45
		$\rightarrow$ $S1_{[f]}$ $S3_{[f]}$	p=0.1
		$\rightarrow$ $S1_{[f]}$ $S4_{[f]}$	p=0.1
		$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$	p=0.1
		$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$ $S4_{[f]}$	p=0.1
		$\rightarrow$ $S1_{[f]}$	p=0.05
		$\rightarrow$ $S2_{[f]}$	p=0.05
		$\rightarrow$ λ	p=0.05
24.	$G3_{[f]}$	$\rightarrow$ $S5_{[f]}$ $S7_{[f]}$	p=0.45
		$\rightarrow$ $S5_{[f]}$ $S6_{[f]}$	p=0.1
		$\rightarrow$ $S6_{[f]}$ $S7_{[f]}$	p=0.1
		$\rightarrow$ $S4_{[f]}$ $S5_{[f]}$	p=0.1
		$\rightarrow$ $S7_{[f]}$ $S8_{[f]}$	p=0.1
		$\rightarrow$ $S5_{[f]}$	p=0.05
		$\rightarrow$ $S7_{[f]}$	p=0.05
		$\rightarrow$ λ	p=0.05
25.	$G7_{[f]}$	$\rightarrow$ $S11_{[f]}$ $S15_{[f]}$	p=0.55
		$\rightarrow$ $S11_{[f]}$ $S16_{[f]}$	p=0.43
		$\rightarrow$ λ	p=0.02
26.	$G8_{[f]}$	$\rightarrow$ $S13_{[f]}$ $S18_{[f]}$	p=0.55
		$\rightarrow$ $S13_{[f]}$ $S17_{[f]}$	p=0.1
		$\rightarrow$ $S14_{[f]}$ $S18_{[f]}$	p=0.1
		$\rightarrow$ $S15_{[f]}$ $S17_{[f]}$	p=0.1
		$\rightarrow$ $S13_{[f]}$	p=0.05
		$\rightarrow$ $S18_{[f]}$	p=0.05
		$\rightarrow$ λ	p=0.05
27.	$G12_{[f]}$	$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$ $S17_{[f]}$	p=0.45
		$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$	p=0.25
		$\rightarrow$ $S7_{[f]}$ $S14_{[f]}$ $S18_{[f]}$	p=0.25
		$\rightarrow$ λ	p=0.05
28.	$G\text{-}G1_{[f]}$	$\rightarrow$ $S1_{[f]}$ $S2_{[f]}$	p=0.2
		$\rightarrow$ $S1_{[f]}$ $S3_{[f]}$	p=0.05
		$\rightarrow$ $S1_{[f]}$ $S4_{[f]}$	p=0.05
		$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$	p=0.05
		$\rightarrow$ $S2_{[f]}$ $S3_{[f]}$ $S4_{[f]}$	p=0.05
		$\rightarrow$ $S1_{[f]}$	p=0.2
		$\rightarrow$ $S2_{[f]}$	p=0.2
		$\rightarrow$ λ	p=0.2
29.	$G\text{-}G3_{[f]}$	$\rightarrow$ $S5_{[f]}$ $S7_{[f]}$	p=0.2
		$\rightarrow$ $S5_{[f]}$ $S6_{[f]}$	p=0.05
		$\rightarrow$ $S6_{[f]}$ $S7_{[f]}$	p=0.05
		$\rightarrow$ $S4_{[f]}$ $S5_{[f]}$	p=0.05
		$\rightarrow$ $S7_{[f]}$ $S8_{[f]}$	p=0.05
		$\rightarrow$ $S5_{[f]}$	p=0.2
		$\rightarrow$ $S7_{[f]}$	p=0.15
		$\rightarrow$ λ	p=0.25
30.	$G\text{-}G7_{[f]}$	$\rightarrow$ $S11_{[f]}$ $S15_{[f]}$	p=0.30
		$\rightarrow$ $S11_{[f]}$ $S16_{[f]}$	p=0.30
		$\rightarrow$ λ	p=0.40
31.	$G\text{-}G8_{[f]}$	$\rightarrow$ $S13_{[f]}$ $S18_{[f]}$	p=0.15
		$\rightarrow$ $S13_{[f]}$ $S17_{[f]}$	p=0.05
		$\rightarrow$ $S14_{[f]}$ $S18_{[f]}$	p=0.05
		$\rightarrow$ $S15_{[f]}$ $S17_{[f]}$	p=0.05
		$\rightarrow$ $S13_{[f]}$	p=0.2
		$\rightarrow$ $S18_{[f]}$	p=0.25
		$\rightarrow$ λ	p=0.25
32.	$G\text{-}G12_{[f]}$	$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$ $S17_{[f]}$	p=0.2
		$\rightarrow$ $S6_{[f]}$ $S14_{[f]}$	p=0.2
		$\rightarrow$ $S7_{[f]}$ $S14_{[f]}$ $S18_{[f]}$	p=0.25
		$\rightarrow$ λ	p=0.35
35.	$n_{[f,t]}$	$\rightarrow$ $S1_{[f]}$	p=0.05
		$\rightarrow$ $S2_{[f]}$	p=0.05
...	...	...	...
		$\rightarrow$ $S20_{[f]}$	p=0.05

Figure 1: Grammar Rules for a Vehicle Tracking Domain

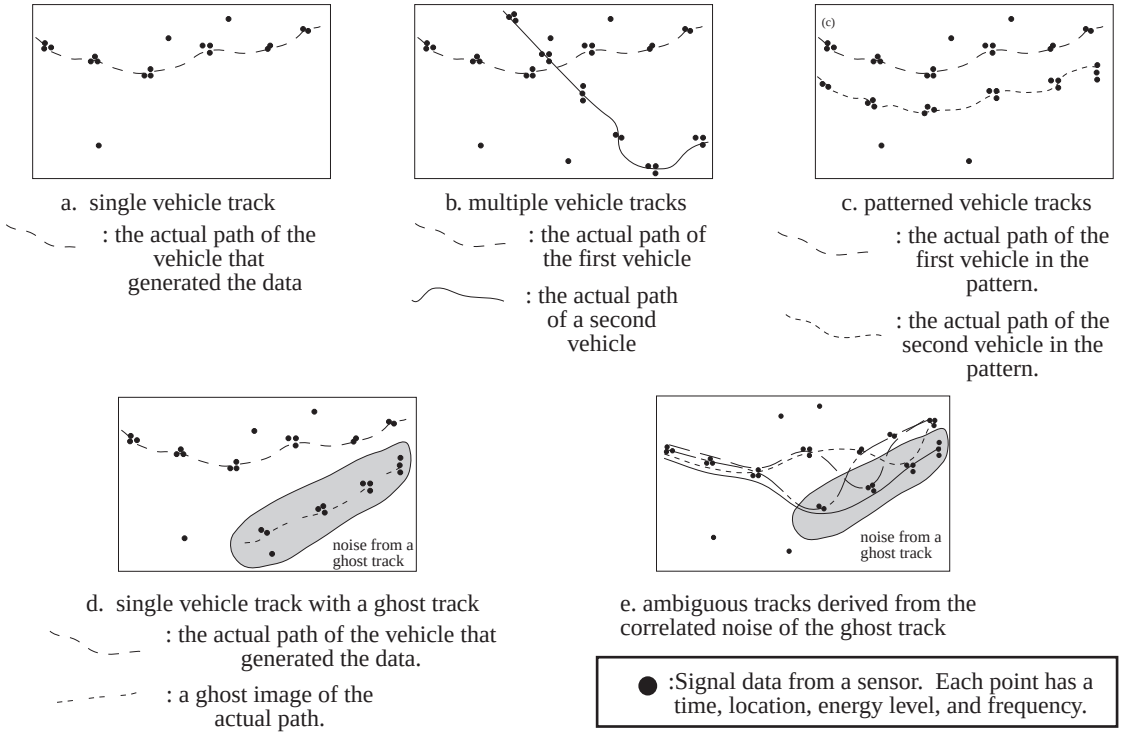


Figure 2: Vehicle Tracking Scenario Examples

rule will not lead to the production of signal data.

Rules 1 through 9 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. Rules 4 through 9 generate the time/vehicle locations for each track[5].

Simple scenario examples are shown in Fig. 2. 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. 2.a. Figure 2.b is an example of multiple tracks in a single scenario. A pattern track is shown in Fig. 2.c. This is contrasted with the ghost track in Fig. 2.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.

2.1 Interacting Phenomena

The 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

[5]Note that these rules are not grounded. They continue to generate data until some predetermined time limit is reached or until the data generated falls outside of the region sensed by the problem solver.

consisting of two tracks with closely related properties. As shown in Figures 2.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 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 and 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 might be considered context-sensitive. The power of the analysis tools presented here is derived in large part from their context-free nature.

2.2 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. We have developed techniques for modeling a variety of different types of noise and a formal definition.

Definition 2.1 *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_1un_2Bn_3vn_4, for u,v $\in$ (V $\cup$ N)*, A $\in$ 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.

Production rules 18 and 35, shown in Fig. 1, are used to generate *random noise*. This represents random phenomena associated with the 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 our simulator, we determine the x and y locations of random noise 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 rule 23,
$G1_{[f]} \rightarrow S2_{[f]}\ S3_{[f]}\ S4_{[f]}$, the extra signal, S4, could be considered noise.

More comprehensive noise structures can be represented at other levels of the grammar in a similar way. For example, ghosting phenomena is 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.

2.3 Correlated and Uncorrelated Noise

In [7], we identified two kinds of noise, correlated and uncorrelated. Correlated noise leads to the generation of additional high-level 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 pruning operators [7].

The complexity of the example grammar is such that it is difficult to identify all correlated and uncorrelated noise by inspection alone. However, the rules associated with the non-terminal n include examples of both. For example, the first LHS of production rule 35: (see Fig. 1) $n_{[f,t]} \rightarrow S1_{[f]}$ 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 last LHS of production rule 35: $n_{[f,t]} \rightarrow S20_{[f]}$ 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. 2.e. In this example, production rule 18 from Fig. 1: $N_{[f,t]} \rightarrow n_{[f,t]}$ $N_{[f,t]}$ 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 an 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.

2.4 Missing Data

Missing data 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. Formally,

Definition 2.2 *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 $\in$ 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.

The second and third RHSs to rule 19 show examples of how missing group level data can be modeled by omitting the appropriate non-terminal. The sixth and seventh RHSs of rule 23 show how missing signal data can be modeled in a similar way.

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 non-terminal.

3 Characterizing Domain Structures

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 [7] 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 in Definition 1.1. 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 occurance 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.

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 sample grammar, 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 correspondance 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 3.1 $D(A) = P(S \vdash^* A)$, *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}). Uncertainty regarding 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. 1), 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, $D(I-Track1) = 1.11 * 0.25 = 0.275$. Similarly, $D(I - Track2) = 0.275$, $D(G - Track1) = 0.11$, $D(G - Track2) = 0.11$, $D(P - Track1) = 0.165$, and $D(P - Track2) = 0.165$.

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 $D(I - Track1) = 2 * 0.25 = 0.5$, $D(I - Track2) = 0.5$, $D(G - Track1) = 0.2$, $D(G - Track2) = 0.2$, $D(P - Track1) = 0.3$, and $D(P - Track2) = 0.3$.

Definition 3.2 $\{RHS(A)\}$ = 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 3.3 $P(b \in \{RHS(A)\}^{\vdash +}) = \sum_{\forall i} P(b \in RHS_i(A)) + \sum_{\forall r'}(P(r' \in \{RHS(A)\}) * P(b \in \{RHS(r')\}^{\vdash +}))$, 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,

$$\{RHS(V2)^{\vdash +}\} = \{G3, G8, G12, S4, S5, S6, S7, S8, S13, S14, S15, S17, S18\} \quad (1)$$

$$P(S4 \in \{RHS(V2)\}^{\vdash +}) = \quad (2)$$

$$P(G3 \in \{RHS(V2)\}) * P(S4 \in \{RHS(G3)\}^{\vdash +}) = 0.65 * 0.1 = 0.065 \quad (3)$$

Definition 3.4 $P(A \vdash^* b) = D(A) * P(b \in \{RHS(A)\}^*), A \in SNT, b \in V \cup N$. Probability that the partial interpretation, b, is generated from full or partial interpretation A, where b is a descendant of A.

Definition 3.5 *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 [7]). 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 3.6 $P(A \cap b) = P(A \vdash^* b) + \sum_{\forall B} P(B \vdash^* b), A, B \in SNT, b \in V \cup N$, 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 3.7 $P(a \cap b) = P(a) * P(b \in \{RHS(a)\}^{\vdash +}), a \in N, b \in V \cup N$. Intersection of domain events a and b, where b is a descendant of a and where a is a nonterminal that is not an SNT.

Definition 3.8 $P(b) = \sum_{\forall A} D(A) * P(b \cap A \mid A), A \in SNT, b \in V \cup N \cup SNT$. 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 3.9 *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 occurance 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.

Definition 3.10 *MARKER(*$a, b) = \frac{P(a \cap b)}{P(a)}$, where a is a nonterminal and b is any terminal or nonterminal.

Knowledge of markers can be used to design a problem solving architecture and to construct focusing mechanisms in a dynamic control strategy. 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, the strongest marker at the group level for an individual component of an I-Track1 (i.e., a vehicle location, V1) is G1. Specifically,

$$MARKER(V1, G1) = P(V1 \cap G1)/P(V1) = \tag{4}$$

$$(P(V1) * P(G1 \in \{RHS(V1)\}^{\vdash +}))/P(V1) = P(G1 \in \{RHS(V1)\}^{\vdash +}) = 0.95. \tag{5}$$

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 [2]. 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.

$$MARKER(G8, S15) = P(G8 \cap S15)/P(G8) = \quad (6)$$

$$(P(G8) * P(S15 \in \{RHS(G8)\}^{\vdash +}))/P(G8) = P(S15 \in \{RHS(G8)\}^{\vdash +}) = 0.1. \quad (7)$$

A formal definition of the differentiator relationship can be given as:

Definition 3.11 *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 [2]. In control algorithms, differentiators can be used to focus problem solving activity [3].

In the vehicle tracking grammar, S1, S2, and S3 are strong differentiators for the event V1 (a vehicle location of type 1). Specifically,

$$DIFF(V1, S1) = P(V1 \cap S1)/P(S1) = \quad (8)$$

$$(P(V1) * P(S1 \in \{RHS(V1)\}^{\vdash +}))/P(S1) = 0.55 * 0.70/0.39 = 0.99. \quad (9)$$

(Note that S1 is also generated by production rule 35, and, consequently, $P(S1)$is greater than the probability of S1 being derinved 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 [1]. 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 paper, 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.

4 Conclusion

The techniques presented in this paper for formally modeling interpretation domains represent a necessary step toward developing a basis for explaining, predicting, and designing the performance of sophisticated knowledge-based problem solvers. By explicitly modeling a problem domain, the IDP$_G$ formalism can be used to experimentally test the effects of alternative control strategies. For example, a specific strategy can be applied in different domains by using slightly altered versions of an IDP$_G$ grammar. This could be done to identify the characteristics of domains in which the control strategy is most effective. Alternatively, different control strategies can be compared and contrasted statistically by using an IDP$_G$ grammar to generate numerous experimental problem instances.

More importantly, by explicitly specifying a problem domain, IDP$_G$ formalism makes it possible to analyze its characteristics in ways that can be used to design control strategies. For example, general domain structures such as differentiators and markers can be identified and used to design meta-operators. These structures represent expectations that are derived from statistical abstractions of a domain.

References

[1] 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.

[2] 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.

[3] 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.

[4] King Sun Fu. *Syntactic Pattern Recognition and Applications*. PH, 1982.

[5] 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.

[6] 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.

[7] 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.