Probabilistic Grammars for Plan Recognition

by

David V. Pynadath

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1999

Doctoral Committee:

Associate Professor Michael P. Wellman, Chair
Associate Professor Edmund H. Durfee
Professor John E. Laird
Professor William C. Rounds
Professor Demosthenis Teneketzis

For my parents

ACKNOWLEDGEMENTS

Thanks to Profs. Edmund H. Durfee, John E. Laird, William C. Rounds, and Demosthenis Teneketzis for their many comments and suggestions on this dissertation. Thanks also to all of the members of the Decision Machine Group for all of their feedback at various stages of the research. Thanks to my various office-mates over the years for not overly straining our limited computing resources, and special thanks to Daniel Berwick for providing his stereo system to enhance the research environment. And very special thanks go to Prof. Michael P. Wellman as my research advisor, co-author, etc. for all of his ideas, words, and timely nodding.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDICES

APPENDIX

CHAPTER 1

Introduction

A decision maker operating in the presence of a planning agent must often try to determine the plan of action driving the agent's behavior, based on partial observation of behavior up to the current time. Deriving the underlying plan can be useful for many purposes—predicting the agent's future behavior, interpreting its past behavior, or generating actions designed to influence the behavior itself. Researchers in AI have studied this problem of *plan recognition* for several kinds of tasks, including discourse analysis [18], user modeling [23], traffic monitoring [37], collaborative planning [20], and adversarial planning [1].

Existing plan recognition approaches illuminate several key issues in the knowledge representation and inference subtasks of plan recognition. Modeling the uncertainty inherent in most planning domains provides one of the most difficult challenges to a recognizer. Many approaches use representational languages (e.g., first-order logic) where there is no explicit accounting of the relative likelihood of plan choices or effects. In such instances, the inference mechanism must use heuristics to distinguish between equally possible explanations that differ in plausibility. However, if the representation of the plan domain included an explicit probabilistic model, then the inference mechanism could have a sound, decision-theoretic basis for choosing among the candidate explanations, as well as for a recognizing agent's overall decision process.

The most comprehensive probabilistic approach to plan recognition constructs Bayesian networks to represent the relationships among observed events [9]. The representational language has very few restrictions on the types of allowable relationships, but this generality leads to impractical probability distributions over the entire problem space, so each particular set of observations requires the generation of a new network. An alternative probabilistic approach [21] restricts the representational language, but creates Bayesian

networks over the entire problem space. The restricted representation leads to practical networks, but limits the applicability of the method. A more desirable representation must be more powerful than this restricted language, but without running into the complexity concerns of the more general approach.

The plan recognition framework proposed in Chapter 2 starts from a causal theory of the agent's decision process and of the world dynamics. The recognizer bases its conclusions on its uncertain *a priori* knowledge about the agent's mental state, the world state, and the world's dynamics, which can be summarized (at least in principle) by a probability distribution. It then makes partial observations about the world, and uses this evidence to infer properties of the agent and its plan. However, if we allow arbitrary relationships within the planning domain, the dependency structure of the corresponding distribution can become arbitrarily complex, rendering it impractical for use in real-world problems. We can instead make certain assumptions of independence among elements of our model. Such assumptions restrict the possible relationships that we can represent, limiting in turn the class of problem domains that fit within the model. However, if we choose the independence assumptions wisely, then the model will still cover a significant class of domains, while providing the benefit of a reduced dependency structure that supports operational network inference.

Pattern recognition research provides one possible source for such models. Plans are descriptions of action *patterns*, and therefore any general pattern recognition technique is automatically a plan recognition technique for the class of plans corresponding to the class of patterns associated with the given technique. Grammatical representations provide a potential language for specifying pattern generation processes, as well as providing inference algorithms for reasoning about those processes. If we can model a given plan generation process within the grammatical model's independence assumptions, restricting plan relationships to be in the form of productions, then we can use the inference algorithms to reason about particular plan instantiations. For instance, existing algorithms can translate domain rules of a certain form into a corresponding context-free grammar [43, 35]. We can then treat a sequence of observed actions as a string in the context-free language and use standard parsing algorithms to answer a wide range of queries. However, even though these grammatical approaches are suitable for a wide range of problem domains, the lack of an explicit model of uncertainty and the context-free nature of the grammars limits their applicability.

We can remedy the former by associating probabilities with the restricted rules allowed by these methods. We can then perform an analogous transformation, accounting for the probabilities as well, to produce a *probabilistic* context-free grammar (PCFG) representing a distribution over possible action sequences. Existing PCFG parsing algorithms answer a restricted set of queries, but Chapter 3 presents algorithms to take such a grammar and generate a Bayesian network capable of producing any conditional probabilities of interest. Therefore, once we specify the problem domain within the PCFG model, we can answer most classes of potential plan recognition queries.

However, the probabilistic grammars produced in such a manner are still context-free, severely restricting the problem domains for which the methods are applicable. Chapter 4 describes the types of context sensitivity that many plan recognition domains require, as well as methods that adapt the PCFG algorithms from Chapter 3 to handle these extra needs. To simplify domain specification, we must transform these individual modifications into a unified representational language. A probabilistic, *context-sensitive* grammar (PCSG), defined in Chapter 4 is one possible language, but the generality of the model prevents the development of efficient inference algorithms.

Chapter 5 defines a more restricted model, the probabilistic *state-dependent* grammar (PSDG), which adds an explicit state model to an underlying PCFG model of plan selection. The grammatical structure can still represent the plan/subplan hierarchy through abstraction and decomposition productions. The state model captures the dependence of plan selection on the planning context, including the agent's beliefs about the environment and its preferences over outcomes. The state model also represents the effects of the agent's planning choices on future states (and, subsequently, on future planning choices).

As a generative model of planning, the PSDG model makes stronger assumptions of independence than more general plan recognition approaches. However, the production structure introduces a weak form of conditional independence, sufficient to support compact representations of intermediate plan states—representations more compact than the Bayesian networks used in most probabilistic approaches. Chapter 5 presents algorithms that generate Bayesian network representations of a given PSDG. However, PSDGs often fail to exhibit the conditional independence properties that network inference algorithms require for efficient inference. Chapter 5 also presents specialized inference algorithms that exploit the weaker independence property that PSDGs do exhibit. These algorithms perform inference more efficiently than the Bayesian network algorithms.

Chapter 6 compares the PSDG model with existing representation languages used in plan recognition and other related domains. The specific comparisons illustrate the representational power that the PSDG language sacrifices for its more efficient inference. PSDG domain specifications for traffic monitoring and air combat demonstrate the utility of the model in two plan recognition applications. Chapter 7 further analyzes the contributions of the model, as well as potential extensions to address its limitations.

CHAPTER 2

Plan Recognition

2.1 Existing Approaches to Plan Recognition

The problem of plan recognition is to induce the plan of action driving an agent's behavior, based on partial observation of its behavior up to the current time. We assume that this behavior is a product of an underlying planning process. As the observed agent begins planning, it has a particular mental state, consisting of its preferences (e.g., goals), beliefs (e.g., about the state of its environment), and capabilities (e.g., available actions). We assume the actual planning process to be some rational procedure for generating the plan expected to best satisfy the agent's preferences based on its beliefs, subject to its capabilities. This plan then determines (perhaps with some uncertainty) the actions taken by the agent in the world.

In general, the recognizer observes only the (possibly noisy) effects of some of these actions on the environment, though in some problem domains (e.g., user modeling) it may observe the actions directly. The recognizer uses these observations, in whatever form, to generate hypotheses about which top-level plan or intermediate subplans the agent has selected, or which low-level actions it will perform in the future. The resulting candidates, as well as possible evaluations of their plausibilities, form the basis for decisions on potential interactions with the observed agent. The standard definition of the plan recognition problem covers only the generation of hypotheses and evaluations, but we must consider the ultimate decision-making problem in the design process, since the intended use of the recognizer's output can play a large role in the choice of solution methods.

2.1.1 Event Hierarchies

Kautz's *event hierarchy* formalism [26] is one of the most famous and most comprehensive frameworks for performing plan recognition. The hierarchy uses first-order logic to represent two types of plan relationships: specialization and decomposition. Specialization rules form the basis for an abstraction hierarchy arranging plan classes according to their generality. Decomposition rules specify type restrictions on the steps performed in plan execution and on the interrelationships among these steps and their effects. These rules, as well as general background knowledge, form the knowledge base for the problem domain. A circumscription process transforms this knowledge base into a set of rules sufficient for deductive plan inference.

For instance, consider the example problem of a driver on the highway, trying to predict the actions of the other drivers. Since these actions are normally limited to a small set of maneuvers (e.g. lane changes, passing, exiting), recognition of a driver's maneuvering plan would greatly assist in the prediction of future actions. If we label the driver's top-level plan Drive, then we can represent the various possible maneuvers with abstraction rules like the following:

$$\begin{aligned} \forall x.\mathsf{Left}(x) &\Rightarrow \mathsf{Drive}(x) \\ \forall x.\mathsf{Right}(x) &\Rightarrow \mathsf{Drive}(x) \\ \forall x.\mathsf{Pass}(x) &\Rightarrow \mathsf{Drive}(x) \\ \forall x.\mathsf{Exit}(x) &\Rightarrow \mathsf{Drive}(x) \end{aligned} \tag{2.1}$$

We can also specify subplan sequences through decomposition rules. For instance, a driver can pass on the left by first performing a left lane change, followed by a right lane change after overtaking the obstructing car. To represent this execution within an event hierarchy, we can write the following decomposition rule:

$$\forall x.\mathsf{Pass\text{-}on\text{-}Left}(x) \Rightarrow \mathsf{Left}(S(1,x)) \wedge \mathsf{Right}(S(2,x)) \wedge \kappa(x) \tag{2.2}$$

Since decomposition rules are implications, we can consistently write only one for each plan. Therefore, we create the new event predicate Pass-on-Left, a specialization of Pass. A second possible specialization, Pass-on-Right, would have an analogous decomposition. The term $S(i,x)$ denotes subevent i of event x, where the index value does *not* indicate any ordering information. Such temporal relationships appear in the κ predicate, which

specifies all of the constraints on the decomposition. In this example, κ includes a total order on the steps, but, in general, the constraint may be only a partial order.

There are additional constraints specifying dependencies among the subplans and the state of the world. For instance, κ includes the requirement that the right lane change begins only when the driver performing the pass has moved beyond the car being passed. In general, these constraints can be arbitrarily complex first-order expressions, allowing us to specify very general relationships among subplans, as well as between subplans and world states. The resulting knowledge base represents a hierarchy of plan/subplan event types, as well as general knowledge about the planning agent's environment.

If we add certain closed-world assumptions and circumscribe the knowledge base, we can use deductive inference to reason from a set of observations to hypotheses of possible top-level plans, future actions, etc. For instance, if we observe a driver perform a left lane change followed by a right lane change, the recognizer would include the Pass-on-Left subplan of a Pass maneuver as a possible explanation.

2.1.2 Event Grammars

However, the time complexity of general deductive inference algorithms makes this approach practical for only very small knowledge bases. Fortunately, we can reduce the complexity of inference by restricting the representational language and using specialized algorithms to answer queries. Lin and Goebel [31] restrict the constraint language, permitting use of a faster, specialized message-passing recognition algorithm. Alternatively, if we assume that the agent always executes its subplans and actions sequentially, then we can view the actions we observe as a string in a language. In addition, if there are no constraints among subplans in the decomposition rules, then we can view the action sequences as members of a context-free language [43]. Rules (2.1) and (2.2) become productions of the following form:

$$\begin{aligned}
\text{Drive} &\rightarrow \text{Left Drive} \\
\text{Drive} &\rightarrow \text{Right Drive} \\
\text{Drive} &\rightarrow \text{Pass Drive} \\
\text{Drive} &\rightarrow \text{Exit} \\
\text{Pass} &\rightarrow \text{Left Right} \\
\text{Pass} &\rightarrow \text{Right Left}
\end{aligned} \tag{2.3}$$

The intermediate Pass-on-Left plan is no longer necessary because we can specify multiple productions to expand a single plan. When we have such a grammar, we can use standard parsing algorithms to process a partial string of actions and decide what possible high-level plans are present and which actions may occur in the future. For instance, if the recognizer observes the sequence Left Right Exit, a parsing algorithm could determine that a single pass or two separate lane changes are the possible explanations for the observations. The parsing algorithms are much more efficient than general deductive inference, but this greater efficiency comes at the cost of requiring a total order over subplan decompositions. The event grammar is also incapable of capturing the constraints imposed by the external world.

2.1.3 Heuristic Methods

With both event hierarchies and event grammars, the response to a query will often be a set of candidate hypotheses, as we see in the example from the driving event grammar. Unfortunately, there is no basis within the original domain specification for distinguishing among the possible explanations. As a possible basis for disambiguation, we could augment the specification with heuristic rules to support such preferences. For instance, Kautz's approach prefers explanations that involve a single top-level plan instead of combinations of multiple plans. The algorithm of Lin and Goebel prefers plan scenarios that are more general. In the parsing example, such a heuristic rule would favor the Pass hypothesis over the successive lane change explanation.

In the air combat domain (described in more detail in Section 6.2.2), a fighter pilot has little time to perform the necessary recognition tasks, so he cannot spend time exploring the different alternatives. Therefore, the agent tracking work [41] that addresses this recognition problem uses heuristic methods to disambiguate multiple hypotheses. The recognizer maintains only a single hypothesis, choosing the candidate that, if correct, would have the worst consequences for the recognizing pilot. This heuristic prevents the pilot from ignoring a possibly dire outcome of a rejected hypothesis.

However, with this heuristic, as with most, we can find problem instances where it produces suboptimal behavior. If the recognizing driver observes another car operating directly to its left, the direst outcome corresponds to the accident that would take place if the observed car executed a Left action and moved into the recognizer's lane. A heuristic favoring prediction and explanation candidates with dire outcomes would compel the recognizing driver toward actions that are excessively defensive, because the probability that

the observed car would execute such a Left action is very low. Of course, we could add an additional heuristic to handle this specific problem, as well as any other similar situations that may arise. Many plan recognition approaches aimed at solving a particular problem domain use such heuristic rules to choose among the recognizer's hypotheses. Unfortunately, it is unlikely that any such ad hoc set of rules will be sufficient for a wide range of problem domains.

2.1.4 Probabilistic Methods

The recognizing agent could instead compute a probability distribution over all the candidates. It could then use this distribution as the input of a decision-theoretic procedure for choosing its actions, allowing the recognizer to distinguish among equally possible, but unequally plausible explanations for the observed activity. In the traffic example, the recognizing driver would most likely derive very little expected utility from responding to a possible Left action by the observed car. A decision procedure that considered the probability of such an act, as well as the large negative utility of an accident, would suggest an evasive maneuver only if the probability of the act reached some threshold indicating an abnormal situation.

The most comprehensive probabilistic approach to plan recognition [9] constructs Bayesian networks representing the relationships among observed events. The work is intended as an aid in story understanding, so there are very few restrictions on the types of allowable relationships. Each random variable represents the occurrence of an event of a certain type, and the links among the variables represent the relationships among event types. These relationships could be very general, allowing partial orders, arbitrary constraints on subplan effects, etc.

Because of the generality of the representation, it would be impractical to generate a Bayesian network representing the entire distribution. The networks are instead created for only a particular set of observed events. A generated network would contain variables corresponding to the classifications of these observations and other variables representing possible unobserved events, as determined by the knowledge elements related to the observed events. We can use the network to compute the probability distribution over these unobserved events, but if we observe new events, then we may have to generate a new network before answering any more queries. Since the time complexity of the generation and compilation of a Bayesian network dwarfs that of evidence propagation, this approach can

be inappropriate for domains where we have to answer queries while processing a stream of observations.

The recognition model of Carberry [6], based on the Dempster-Shafer theory of evidential reasoning instead of Bayesian techniques, takes a similar approach by using threshold plausibility and difference levels of belief to distinguish among competing hypotheses. An alternative probabilistic approach [21] restricts the representational language, but creates Bayesian networks over the entire problem space, for those problems that fit within the language. However, the language restrictions do not allow adequate modeling of the planning process, so the networks cannot always represent the effects of world state on plan selection. Therefore, although the resulting networks support feasible inference, the range of problems that the mechanism supports is too limited.

2.2 Bayesian Plan Recognition Framework

In many plan recognition domains, the recognizing agent would greatly benefit from a probabilistic approach, but such an approach requires a representation language powerful enough to capture the structure of the domain, while still supporting practical inference. This section proposes a general framework for a Bayesian approach to plan recognition, within which we can analyze the types of dependencies present in most problem domains. To perform plan recognition tasks within this framework, we specify a causal planning model along the general structure of Figure 2.1 and use it to support evidential reasoning from observations to plan hypotheses. That diagram can be viewed as a Bayesian network, where the limited connections among the nodes reflect the dependency structure of our generic planning model.

To illustrate the general model, this section presents a hand-coded Bayesian network, following the dependency structure of Figure 2.1, of the maneuvers of a single car. To make the network operational, we replace each component of the model with a subnetwork that captures intermediate structure. We can then use this model to identify the current maneuver of an observed car and/or predict future actions, given only partial information. The subnetwork descriptions below first present the general modeling issues for the corresponding component and then provide a specific instantiation, in the form of a Bayesian subnetwork, for this specific traffic domain.

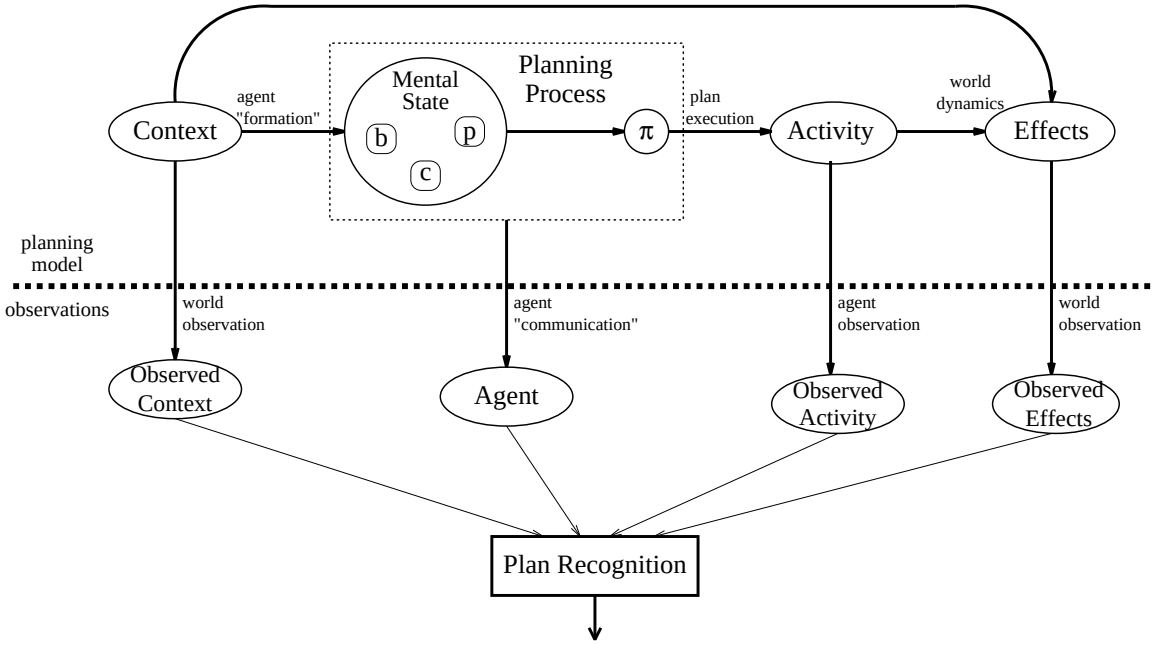

Figure 2.1: Plan recognition framework.

2.2.1 Context

The causal model begins with the initial world state. We must include all possibly observable events that are relevant to *formation*—the process by which the agent's mental state is affected by the world. By including these events, the recognition procedure can take advantage of partial information about the agent's mental state. Note that even though the initial world state model may itself include inaccessible variables, the context subnetwork includes only those which are observable. However, we may wish to simplify the network by providing more compact intermediate results derived from inaccessible variables.

One of the motivations for maintaining a separate initial-state subnetwork is to distinguish between our contextual observations and those of the agent. Therefore, we may have an unobservable node representing an aspect of the world state accessible to the agent, and an observable node representing a related feature accessible to us. The dependency between these nodes is essentially a sensor model. If we are fortunate enough to have perfect sensors, then the context variables become redundant, since they will simply echo the values of the actual variables, and can be eliminated.

In this model, the initial world state is defined as causally prior to all agent behavior. Therefore, the corresponding random variables can have links only from other such variables,

representing dependencies within the state. Any dependency links connecting a node from the initial state to any node outside this subnetwork must be directed to the outside node.

This treatment of context differs from the work of Huber et al. [21], where the initial situation depends on the agent's mental state and not the other way around as it is here. This was possible given the planning model employed in that work, that of the Procedural Reasoning System (PRS) [22]. In the PRS model, plan selection is a function of current goal and situation. Because these context variables have no predecessors or substructure, the direction of links can be reversed without changing the rest of the dependency structure. However, the agent's mental state considered here may be more complex, especially in terms of its preference structure. Even if the agent has only simple goals, there are potential interactions among the goals that could affect the planning process. Hopefully, by following the causal structure in creating the network and placing the context prior to the plan, we can represent these interactions without greatly complicating the dependency structure.

In the traffic domain, the driver must consider several aspects of the initial world state in rationally choosing a plan. First of all, the current position and speed of the car are important factors, and we assume that both are observable, to the driver as well as to us. We also assume perfect sensors, but an extension to incorporate sensor noise is straightforward, as described above. The random variables x position and y position of Figure 2.2 represent the car's lane position and distance from the highway's start, respectively. The driver can be in one of three lanes or may be off the highway, either preparing to enter or having just exited. The random variable y speed, denoting the car's speed, initially depends on the current node, since the farther left the lane, the faster the car is usually traveling.

We can also observe the presence of other cars around the driver of interest, who must consider them in choosing a maneuver. For instance, if there is a car blocking the driver's front, then a passing maneuver is more likely. We can observe any cars to the driver's immediate front, back, left, and right, as well as in the four diagonal directions. In the Bayesian network, the Boolean random variable left clr? represents the presence of any car to the immediate left of the driver. There are similar variables for the right, front, and back, as well as the four diagonal directions. The variables indexed t0 in the first column of nodes in Figure 2.2 constitute the context subnetwork.

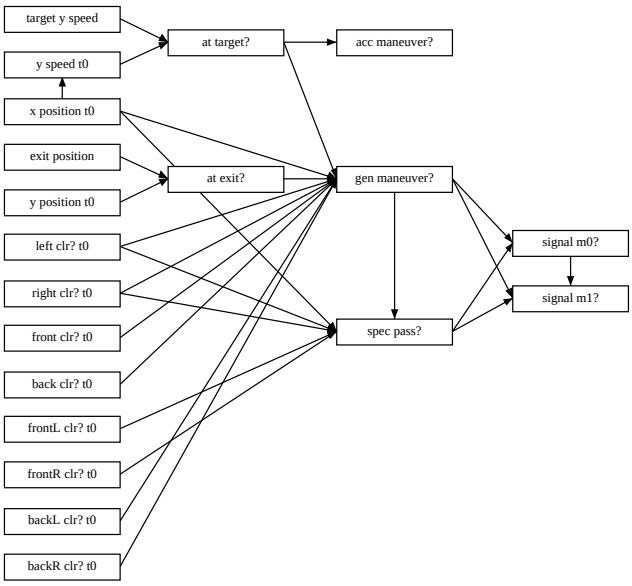

Figure 2.2: Planning process subnetwork.

2.2.2 Mental State

The plan-recognition framework should accommodate all possible information about the agent's plan selection process, beginning with its mental state. We can break down an agent's mental state into three distinct components:

Beliefs. The agent's knowledge of the state of the world and its dynamics. Beliefs may be incomplete, uncertain, or incorrect.

Preferences. The agent's desires about the world. These may be simple goals, or arbitrarily graded degrees of utility.

Capabilities. The agent's self-model of its available actions. Strictly speaking, this should be *knowledge of* capabilities, but we stick to the more concise term.

The representation of the agent's beliefs about the world state must include random variables for all aspects of the context that the agent can observe and that factor into its decision-making. There may be some agent beliefs that are independent of any real-world variable. Unless we can observe these beliefs(perhaps through communication with the agent), there is no advantage in adding the corresponding random variables. Instead, we can fold the uncertainty in these beliefs into the plan component. However, agent beliefs will typically depend on the some aspect of the actual state of the world, although we can model the agent as being arbitrarily uncertain or deluded. As mentioned in Section 2.2.1,

this dependency represents the imperfection and/or incompleteness of sensors. If the agent's sensors were perfect, then we could eliminate the nodes for the agent's belief variables, as they would take on the same values as the context variables.

The agent's knowledge of its capabilities is usually independent of the world state, as are its preferences in most cases. Simple goals can be represented as separate Boolean variables, though it may be useful to combine a set of mutually exclusive variables into a single variable with several possible values. More complex preference structures will require more complex subnetwork structures. The agent's capabilities can be represented in a similar fashion.

The model of agent formation is greatly simplified in our traffic domain. Because of our assumption of perfect sensors, the driver's beliefs about the world correspond to the actual values in our simplified model. In addition, the agent's beliefs about its capabilities are not represented explicitly in our traffic network. Instead, the driver is assumed to know all of the possible plans (as described in Section 2.2.3). The planning process also assumes that the driver has complete knowledge of how the plans can best satisfy its preferences in the current context. Thus the plan selection mechanism implicitly represents the driver's beliefs about its capabilities.

We model the driver preferences with two goals. First, a driver has the explicit goal of getting from one exit to another, though the intended exits are unknown to an external observer. The random variable exit position in Figure 2.2 represents the driver's desired exit. All of the possible exit positions are farther along the highway than the values of y position. If this were not the case, then the current position would provide evidence that the desired exit is probably not one that has been passed. Therefore, there would be a dependency, but to simplify the network, we make the sets of y and exit positions disjoint.

Second, there may be some constraint on the travel time between these exits, or the driver might have some target speed which is preferred for the duration of travel. However, we can usually translate the former into a desired speed because of the fixed positions of the exits. Therefore, our model uses only the random variable target y speed in Figure 2.2, with its values clustered around the speed limit. If the car has been on the highway for enough time, then its current speed should provide some clue as to the driver's target speed. We could model this with a link from y speed. On the other hand, if we have been observing the car and its maneuvers for some time, then these past observations should provide more conclusive evidence as to its target speed. Thus, we can make the target speed independent of current speed and encode our past observations in the prior probabilities.

This network also contains the intermediate belief random variables, at exit? and at target?, in the second column of nodes in Figure 2.2. These reflect the driver's belief about the proximity of the desired exit and the desirability of the current speed, respectively. The at exit? variable depends only on the current position and the preferred exit, and is true only when the former is immediately before the latter. The at target? variable depends only on the current and preferred speeds, and its value indicates whether the current speed is too slow, too fast, or just right, with respect to the driver's desired cruising speed.

2.2.3 Planning Process

Plan Variables

The plan component is comprised of random variables collectively representing the current plan. For instance, in Kautz's event hierarchies, there is a taxonomy of plans and actions. The children of a certain plan correspond to possible subplans or actions, while other links indicate necessary components. If our planning model is based on such event hierarchies, we may designate one Boolean variable corresponding to each element in the taxonomy, indicating the presence of the corresponding plan. Or we may combine certain mutually exclusive subplans into a single random variable, which takes on a different value depending on the actual subplan present.

Such hierarchies are based on the subsumption relation, requiring a dependency link from the more general node to the more specific. The conditional probability table can represent the distribution of the specific values, given the general. In particular, because of the subsumption relation, we can set the conditional probability of a child node given that its parent node is false to zero.

In the traffic domain, we can classify driving maneuvers according to the lane changes involved. The simplest plan is to simply continue driving in the same lane. At the next level of complexity, a driver can shift one lane to the left or right. We consider entering and exiting the highway as specific instances of these one-lane shifts. The driver could also shift two lanes to the left or right, where this could again involve entering or exiting the highway. As a final option, the driver may choose a passing maneuver, which we view as two successive lane shifts of opposite direction. In Figure 2.2, the variable gen maneuver represents the general driving maneuver and takes on a value corresponding to the chosen plan.

We can also classify driving plans according to the acceleration. Depending on the

current and desired speed, a driver may decide to speed up, slow down, or maintain current speed, indicated by the value of the variable acc maneuver of Figure 2.2. The acceleration maneuver depends on the lane maneuver if we do not consider the plan selection mechanism. For instance, a deceleration is more likely as a part of a right lane change plan than as a part of a plan to pass. However, the two variables are independent given the initial context, as indicated in the network.

The variable spec pass in Figure 2.2 indicates the direction of the pass, if one is taking place. Since passing in a specific direction is a subplan of the general passing maneuver which gen maneuver can represent, this is an example of the subsumption relation found in event hierarchies. If the driver decides to pass, there are the options of passing on the left and passing on the right. And even if the driver chooses to pass, there may be cars blocking both lanes, forcing the driver to wait for another opportunity to pass. This variable clearly depends on gen maneuver, since the more general passing maneuver is its parent and the conditional probability table represents a subsumption relation as described above. In other words, if a passing maneuver is *not* chosen, then spec pass will be neither pass on left nor pass on right.

Plan Selection

Links from the agent's mental state into the plan component represent the agent's planning process. For hierarchical planning, we start with the most general plan nodes and proceeding to the most specific, determine which aspects of the mental state influence the agent's choice. For instance, suppose the agent's decision-making procedure consists of a set of condition-action rules. Then, any plan choices in the action portion of a rule depend on all of the context variables that appear in the conditions of the rule. By connecting only parts of the mental state relevant to particular choices, we keep the dependency structure as simple as possible.

We must then specify the conditional probabilities of the plan variables given the relevant aspects of the agent's mental state. If the agent is a deterministic planner, then the conditional probability given a particular mental state instantiation will be 1 for a single instantiation of the plan subnetwork and 0 for all others. For nondeterministic planners, we must determine the conditional probabilities from whatever agent model we have.

If in fact we have no opportunity to observe anything about the initial world state or the agent's mental state, then we may collapse the initial state and mental state subnetworks

into prior probabilities for the top-level plan variables. The plan recognition networks (PRNs) of Charniak and Goldman [9] use such priors to model the agent's plan selection process. These prior probabilities represent the same distribution as the explicit planning process subnetwork, but since the initial nodes are unobservable, we can merge the nodes into the plan subnetwork without losing information.

We can now model a driver's plan selection with some reliability. In our Bayesian network, the conditional probability table must specify the likelihood of certain maneuvers under every possible combination of world situation and driver mental state. Under most situations, there will be one maneuver that is clearly preferable, though there is still uncertainty. For example, suppose that the driver is currently traveling below the desired speed and that there is another car directly in front while the lane to the left is clear. Then it is likely that driver will pass the car on the left. The complete plan selection subnetwork is shown in Figure 2.2. This model of the driver's decision process is based in part on the driving model underlying the BATmobile (Bayesian Automated Taxi) project, described by Forbes, et al. [15].

The acceleration maneuver depends only on the preferability of the current speed. Thus the sole link to acc maneuver is from at target?. If the driver is at the target speed, then the current speed will be maintained. If the current speed is too low, then the driver will choose an acceleration maneuver. Likewise, if the current speed is too fast, then a deceleration maneuver will be chosen.

The lane change maneuver also depends on the preferability of the current speed. For instance, a car traveling at its target speed is unlikely to change lanes. However, there are other factors in the initial world state to consider. Obviously, the current lane is important, since a car in the leftmost lane cannot change lanes to the left. In addition, the driver will consider any cars to the front or back. If there is a car blocking the front and the driver's current speed is too low, then a simple acceleration could cause a collision. The driver may instead choose to change lanes to the left. But a decision to change lanes must also consider the presence of cars to the driver's left or right, or any cars coming up from the back left or right. The links to the gen maneuver node represent these dependencies.

If the driver decides to pass, a direction must be chosen. Passing on the left is preferable to passing on the right, but the current situation may not allow it. For instance, any cars to the driver's left or to the front left could block the passing attempt. The same is true on the right side. If enough passing avenues are blocked, then the driver may decide to delay

the passing attempt or to perform the initial lane change and wait to complete the pass.

Agent Communication

Modeling agent communication depends greatly on the specific protocol adopted, and the relationship between the observer and the observed. If a trusted agent directly announces particular aspects of its planning process, then we could simply instantiate the corresponding variables. Other types of communication would require nodes to represent beliefs we attribute to the agent, based on its communication actions. Note that we are not modeling here the planned character of communication acts; to do so we would treat them as we do actions in general.

The only communication allowed in our traffic model is through the driver's turn indicator, which provides a simple mechanism for a driver to announce the intended lane change. The variables signal mx? in the fourth column of Figure 2.2 represent the state of the driver's turn signal during stage x of the maneuver. Clearly, both the general maneuver and the specific direction of any passing attempt influence any signal. For instance, when performing a left lane change, signal m0? is likely to take the value *Left* and signal m1? the value *Off*. Of course, many drivers fail to signal their maneuvers, and sometimes they signal erroneously. These possibilities are considered when determining the conditional probability tables. However, drivers are usually consistent in their signaling habits. For instance, when performing a pass on the left, someone who fails to signal the initial left lane change is unlikely to signal the subsequent right change. The link between the two signal variables represents this consistency.

2.2.4 Plan Execution

Once we have accounted for the agent's plan-generation process, we need to consider the effects of the plan's execution, reflected in the model by the dependency of the activity component on the plan component. This is analogous to links in event hierarchies connecting plans to their component observable actions. In PRS [22, 29], Knowledge Areas (KAs) specify a sequence of actions associated with a plan, corresponding to links from the plan node to corresponding action nodes. Either of these can be cast in Bayesian networks, representing the likelihood of the component's appearance given the plan in the conditional probability table for that node.

The activity subnetwork in the traffic model includes the individual transitions in lane

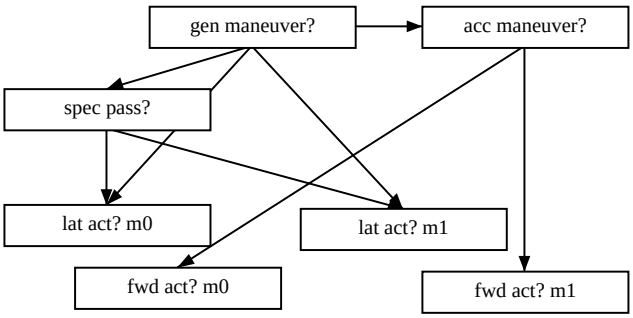

Figure 2.3: Plan execution subnetwork.

and speed, which are completely unobservable. At each step, the driver can change one lane to the left or right, or remain in the same lane. The driver can also increase, decrease, or maintain speed. All of the plans we consider produce a two-step action sequence. For instance, a plan to shift one lane to the left produces a left lane change followed by a "remain in lane" act. The lat act mx variables in Figure 2.3 represent the lane changes at step x, while fwd act mx represents the acceleration at step x.

Our definition of the lane maneuvers completely determines the lane changes of the action sequences. The individual shifts depend on the general lane maneuver, as well as on the specific passing plan, but not on the acceleration maneuver. Likewise, the individual accelerations are independent of the general lane changes and the specific passing maneuvers if given the overall acceleration plan.

2.2.5 World Dynamics

The relationship between the observed and actual actions of the agent is similar to that of the observed and actual world states. If we have perfect sensors, we do not need a separate observed activity subnetwork; otherwise, we have to model sensor noise in the links from the actual nodes. In some cases, the agent's activity is completely inaccessible, though we might still be able to observe effects of this activity. These effects are dictated by the dynamics of our world, which specify how the agent's actions alter the situation. Therefore, we must model how subsequent world states depend on the initial world state and the agent's activity. It is possible that a world state depends on the entire world history, but if the the plan is sufficiently structured (e.g., sequential actions) then we may be able to simplify this dependency. If we express the effects model in accord with standard AI approaches, we can restrict the effects to depend only on background and direct effects and,

given these, to be conditionally independent of the plan itself, as well as further removed activity and indirect effects [44].

We can make effects conditionally independent of future actions and effects simply by ensuring that links never point backward in time, but this could make actions dependent on past world states. So far, we have had links move from plans to activity and from activity to effects, so adding links in the opposite direction would go against the flow in Figure 2.1. If, as described above, the plan is sufficient for determining activity, the current action is conditionally independent of the previous world states given the current plan, as well as the actions performed so far.

Depending on our domain, we may able to make a Markov assumption with respect to activity and the effects. In such cases, the current action would be conditionally independent of actions more than one time step back in the action sequence, given the current plan and the action immediately previous. If the effects have a similar property, they should not depend on any world states or actions more than one step previous. Although this would greatly localize the dependencies, this may not always be possible, depending on the types of observations available and the set of state variables in the model.

Some domains may allow us to make even more simplifying assumptions about the agent's planning process. Some simple agents may fit within a hidden Markov model [39]. In such cases, the agent performs a sequence of actions, each of which depends only on the immediately previous action. Evidence is available in a corresponding sequence, with the observations at a particular stage dependent only on the action at the same stage. These independence assumptions produce a very simple belief network which can still handle the wide range of plan recognition queries, even though the agent no longer has an explicit plan.

Since there is no directly observable activity in the traffic model, most of our inference will come from observed effects. We must now model the dynamics of the traffic world, beginning with the changes in the position and speed of the car. We can view the actions of the driver to be transitions between world states. To simplify the model, we ignore observations taking place while the driver is performing an action. Thus, evidence is available only at the completion of a component action, and there are three stages of observable variables, including the context, as can be seen in Figure 2.4.

Finally, we must define the dependencies of these effects. Most of the observable variables depend on the driver's previous action, as well as their own previous values. For instance, the driver's lane is completely determined if we know what lane change just took place, as

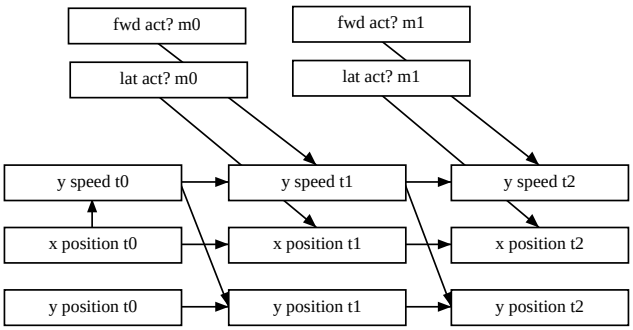

Figure 2.4: Observation subnetwork.

well as the lane value just before the change. Likewise, the driver's speed depends on the previous speed and whatever acceleration action took place, although this is clearly not a deterministic relationship.

The presence of other cars is a bit more complex, due to the driver's movements. For instance, after a left change, a car that was to the front and left is now probably directly in front. But if the driver stays in the same lane, then we must check whether there was a car blocking the front in the previous world state. Therefore, each clearance variable depends on the previous action, as well as all relevant clearance variables from the previous state. To simplify the network, we ignore the presence of other cars in the evidence. We do consider them when modeling plan selection, but since the driver's actions do not directly affect the other drivers' positions, we ignore these effects. As with the context, we assume perfect sensors, so there is no distinction between the actual and observed effects.

2.2.6 Inference with the Traffic Bayesian Network

Once we have created the belief network, we can perform recognition tasks by fixing any observed variables and querying the network about the relevant variables. We receive evidence only about the variables in the bottom half of Figure 2.1, though, as described before, these may correspond exactly to actual variables in the planning model.

Once we fix the values of the known variables in the network, we can propagate the information throughout the network and observe the posterior probabilities at the nodes of interest. For instance, we may be interested in determining the plan chosen by the agent, in which case we would examine the nodes in the plan subnetwork. Alternatively, we can predict future agent activity or effects by examining the probabilities of those variables.

Once we have constructed the entire traffic maneuver network, shown in Figure 2.5,

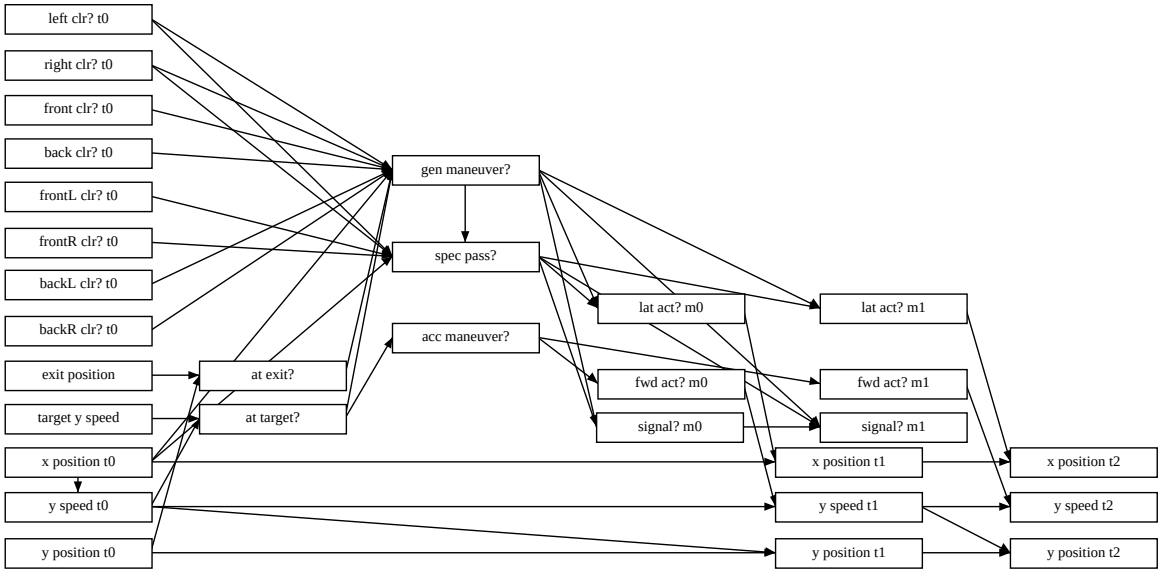

Figure 2.5: Complete Bayesian network for traffic monitoring.

we can handle plan recognition in a wide range of useful driving situations. For instance, suppose that as, our exit approaches, we are preparing to move into the rightmost lane from our current position in the middle lane. However, before we can change lanes, the car behind us in the middle lane moves into the rightmost lane. If it is preparing to pass us, we can simply wait for it to move ahead of us before changing lanes and exiting. If, however, the car is simply changing lanes, or perhaps preparing to exit itself, we can change lanes, slowing down or speeding up as necessary to avoid a collision. To best plan a course of action, we want to compute the probability distribution over the possible explanations.

Thus, in the context, we have observed front clr? t0 to be false and x position t0 to be the middle lane. The only observed effect is that x position t1 is the right lane. If we want to infer the driver's plan, we can examine the gen maneuver? node to see that the posterior probability of a one-lane right shift is 0.64, while that of a pass is 0.35. The former is more plausible since we assume that drivers prefer to pass on the left-hand side, so passing on the right has a relatively low prior probability. The only remaining maneuver with nonzero probability is an exit. All of the other plans have zero probability, since the observed change in lanes violates their definitions.

If we are not interested in the driver's plan, but only in the future lane position, then we can examine the x position t2 node. The posterior probability that the car will still be in the right lane is 0.65, while the probability that it will move to the middle lane is 0.34.

The difference between these beliefs and that of the maneuvers arises from the nature of the passing maneuver. Even if the car decides to pass, it may not be able to do so immediately do to surrounding cars. In such a case, it will remain in its current lane until it can complete the maneuver. Thus, there is a slight probability that the car will stay in the right lane even if the driver has decided to pass.

2.2.7 Representational Language for the General Framework

The traffic application presented above illustrates several aspects of our plan-recognition framework. Our assumption of rationality on the part of the agent allowed us to model the relationship between an agent's plan and its mental state. By creating a probabilistic model of a driver's decision process, we could compute posterior probability distributions that answer the common types of plan recognition queries. In domains where we can generate such Bayesian networks, a recognizing agent can have the desired decision-theoretic basis for interacting with the observed agent.

Although the traffic example is a very specific domain, the general structure of Figure 2.1 is applicable to a broad class of plan-recognition tasks. A language suitable for specifying plan recognition domains should be able to capture the components and dependency structure illustrated by the general framework. This section presents an examination of each of the components and dependencies to analyze the wide variety of requirements of a specification language capable of representing potential problem domains.

The modeling of the planning agent's environment draws on all areas of knowledge representation, since the plan-recognition model makes no restrictions on environmental complexity. We thus face the possibility of an arbitrarily complex collection of features, with arbitrarily complex dependencies among those features. These dependencies may come in the form of deterministic functions (e.g., logical implications, geometric relationships, `at exit?` in the traffic example). In general, these dependencies involve uncertainty, as in the representation of the other drivers in the traffic example, where we used a probability distribution to correctly model the environment. However, the Bayesian network representation presented in this chapter is propositional, so it can represent cars in only the eight positions with corresponding random variables. A more accurate model would include first-order information about other drivers, allowing the recognizer to reason about as many cars as may be relevant to the current planning situation.

The representation of the planning agent's mental state often requires the same capa-

bilities used in the representation of the actual environment. The structure of the agent's beliefs often mirrors the structure of the features of the real world. In addition, agent formation includes all aspects of the derivation of the agent's beliefs, so we must model the agent's sensors and any noise introduced by such sensors. In the traffic example, we could model the driver's preferences by a finite set of goals indicating a target speed and intended exit. However, in many domains, we must model the agent's preferences through a utility function defined over world states.

The representation of the planning process must reflect the vast diversity of plan selection mechanisms. We must often model hierarchies of plans and subplans, representing the agent's choices at different levels of abstraction. We must also represent the decision process generating the agent's choices, whether based on simple condition-action rules or on a complex decision-theoretic analysis of the current situation. Subplans may have preconditions specifying when their execution may begin, termination conditions specifying when their execution must end, and various conditions specifying other changes to their execution.

The agent's planning process may also make different levels of commitment when selecting a particular subplan. In the traffic example, the plan decompositions specified a total order over the subplans, but more general planners may specify only a partial order. In addition, although the traffic example includes concurrent lane change and acceleration maneuvers, in general, we cannot restrict concurrency to a fixed set of orthogonal components.

The modeling of the world dynamics makes the same demands as the modeling of the initial context. The features of the future world state may have an arbitrary dependency on the current world state and the actions taken by the planning agent. We may model certain dynamics through subplan postconditions that specify how the execution of the subplan causes changes in the world. In the traffic example, only the low-level actions cause changes in the world state, but in general, even subplan choices could change the initial context, perhaps by directly altering the agent's mental state.

The representational needs cataloged here reflect the entire span of artificial intelligence research, so it is unreasonable to expect a single language to satisfy the needs required over all plan recognition domains. However, even if there were such a language, capable of representing any model of plan *generation*, it must also support the inversion necessary to answer plan *recognition* queries. Therefore, it is not sufficient that a generative model allows us to reason forward from an initial context through the planning process to the final

world state. We must also be able to reason backwards, from observations taken from the various components, to make inferences about the unobserved components. This need for practical inference in both prediction and explanation of agent behavior severely limits the representational power allowable in our domain specification language.

For instance, even though the Bayesian network representation falls short of all of our representational needs, the scalability of the framework as formulated here is questionable. The framework restricts the dependency structure among the components to follow the limited links of Figure 2.1, but we cannot allow the individual components to be arbitrary Bayesian networks. Otherwise, without any restrictions on the dependency structure within each component's subnetwork, it will be extremely difficult to hand-generate operational network representations for problem domains more complex than the traffic example presented here.

Instead, we need a representational language that captures the general dependency structure of Figure 2.1, but without the full generality of Bayesian networks for component representation. For a sufficiently restricted language, there may be algorithms capable of automatically generating a Bayesian network representation of a domain specification in the language. In such cases, we can still use the Bayesian network inference algorithms, as in Section 2.2.6. However, the automatically generated networks may still be impractical, in which case we would need inference algorithms, specific to the representational language, capable of answering the same plan recognition queries. The work presented here is similar in aim to a proposed plan recognition framework (currently under development) based on probabilistic Horn abduction (PHA) [16].

CHAPTER 3

Probabilistic Context-Free Grammars

Section 2.1.2 discussed how CFGs can model certain plan recognition domains [43]. The production format is a natural representation for plan decompositions when an agent executes its subplans and actions serially. However, like the more general event hierarchy formalism, a grammatical model requires a decision procedure to disambiguate among multiple candidate explanations, parse trees in this case. On the other hand, *probabilistic* context-free grammars (PCFGs) [17, 7] add an explicit stochastic model to the standard CFG model, inducing a probability distribution over parse trees. The resulting distribution supports the decision-theoretic disambiguation required in plan recognition. PCFGs have already provided a useful method for modeling uncertainty in a wide range of structures other than plans, including natural languages [7], programming languages [45], images [11], speech signals [34], and RNA sequences [40]. The discussion in Sections 3.1 and 3.2 appeared previously in a more general pattern recognition setting [38], but Section 3.3 illustrates how we can apply PCFGs to the specific needs of plan recognition.

3.1 Specification of PCFG Language Model

A probabilistic context-free grammar is a tuple $\langle \Sigma, N, S, P \rangle$, where the disjoint sets Σ and N specify the terminal and nonterminal symbols, respectively, with $S \in N$ being the start symbol. P is the set of productions, which take the form $E \rightarrow \xi$ (p), with $E \in N$, $\xi \in (\Sigma \cup N)^+$, and $p = \Pr(E \rightarrow \xi)$, the probability that E will be expanded into the string ξ. The sum of probabilities p over all expansions of a given nonterminal E must be one. The examples in this chapter use the sample grammar (from Charniak [7]) shown in Fig. 3.1.

This definition of the PCFG model prohibits rules of the form $E \rightarrow \varepsilon$, where ε represents

S	$\rightarrow$	np vp	(0.8)	pp	$\rightarrow$	prep np	(1.0)
S	$\rightarrow$	vp	(0.2)	prep	$\rightarrow$	like	(1.0)
np	$\rightarrow$	noun	(0.4)	verb	$\rightarrow$	swat	(0.2)
np	$\rightarrow$	noun pp	(0.4)	verb	$\rightarrow$	flies	(0.4)
np	$\rightarrow$	noun np	(0.2)	verb	$\rightarrow$	like	(0.4)
vp	$\rightarrow$	verb	(0.3)	noun	$\rightarrow$	swat	(0.05)
vp	$\rightarrow$	verb np	(0.3)	noun	$\rightarrow$	flies	(0.45)
vp	$\rightarrow$	verb pp	(0.2)	noun	$\rightarrow$	ants	(0.5)
vp	$\rightarrow$	verb np pp	(0.2)				

Figure 3.1: A probabilistic context-free grammar (from Charniak [7]).

the empty string. However, we can rewrite any PCFG to eliminate such rules and still represent the original distribution [7], as long as we note the probability $\Pr(S \rightarrow \varepsilon)$. For clarity, the algorithm descriptions in this chapter assume $\Pr(S \rightarrow \varepsilon) = 0$, but a negligible amount of additional bookkeeping can correct for any nonzero probability.

The probability of applying a particular production $E \rightarrow \xi$ to an intermediate string is conditionally independent of what productions generated this string, or what productions will be applied to the other symbols in the string, given the presence of E. Therefore, the probability of a given derivation is simply the product of the probabilities of the individual productions involved. We define the *parse tree* representation of each such derivation as for non-probabilistic context-free grammars [19]. The probability of a string in the language is the sum taken over all its possible derivations.

3.1.1 Standard PCFG Algorithms

Since the number of possible derivations grows exponentially with the string's length, direct enumeration would not be computationally viable. Instead, the standard dynamic programming approach used for both probabilistic and non-probabilistic CFGs [24] exploits the common production sequences shared across derivations. The central structure is a table, or *chart*, storing previous results for each subsequence in the input sentence. Each entry in the chart corresponds to a subsequence $x_i \cdots x_{i+j-1}$ of the observation string $x_1 \cdots x_L$. For each symbol E, an entry contains the probability that the corresponding subsequence is derived from that symbol, $\Pr(x_i \cdots x_{i+j-1}|E)$. The index i refers to the position of the subsequence within the entire terminal string, with $i = 1$ indicating the start of the sequence. The index j refers to the length of the subsequence.

The bottom row of the table holds the results for subsequences of length one, and the top entry holds the overall result, $\Pr(x_1 \cdots x_L|S)$, which is the probability of the observed

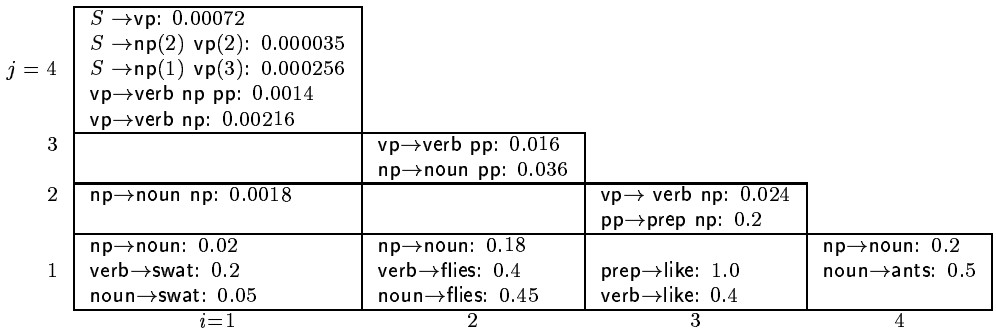

Figure 3.2: Chart for Swat flies like ants.

string. We can compute these probabilities bottom-up, since we know that $\Pr(x_i|E) = 1$ if E is the observed symbol x_i. We can define all other probabilities recursively as the sum, over all productions $E \rightarrow \xi$ (p), of the product $p \cdot \Pr(x_i \cdots x_{i+j-1}|\xi)$. Altering this procedure to take the maximum rather than the sum yields the most probable parse tree for the observed string. Both algorithms require time $O(L^3)$ for a string of length L, ignoring the dependency on the size of the grammar.

To compute the probability of the sentence Swat flies like ants, we would use the algorithm to generate the table shown in Fig. 3.2, after eliminating any unused intermediate entries. There are also separate entries for each production, though this is not necessary if we are interested only in the final sentence probability. In the top entry, there are two listings for the production S →np vp, with different subsequence lengths for the right-hand side symbols. The sum of all probabilities for productions with S on the left-hand side in this entry yields the total sentence probability of 0.001011.

This algorithm is capable of computing any *inside* probability, the probability of a particular string appearing inside the subtree rooted by a particular symbol. We can work top-down in an analogous manner to compute any *outside* probability [7], the probability of a subtree rooted by a particular symbol appearing amid a particular string. Given these probabilities we can compute the probability of any particular nonterminal symbol appearing in the parse tree as the root of a subtree covering some subsequence. For example, in the sentence Swat flies like ants, we can compute the probability that like ants is a prepositional phrase, using a combination of inside and outside probabilities. The Left-to-Right Inside (LRI) algorithm [24] specifies how we can use inside probabilities to obtain the probability of a given initial subsequence, such as the probability of a sentence (of any length) beginning with the words Swat flies. Furthermore, we can use such initial subsequence probabilities

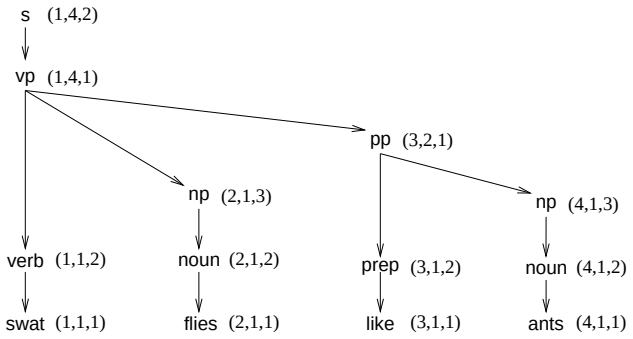

Figure 3.3: Parse tree for Swat flies like ants**, with** (i, j, k) **indices labeled.**

to compute the conditional probability of the next terminal symbol given a prefix string.

3.1.2 Indexing Parse Trees

Yet other conceivable queries are not covered by existing algorithms, or answerable via straightforward manipulations of inside and outside probabilities. For example, given observations of arbitrary partial strings, it is unclear how to exploit the standard chart directly. Similarly, we are unaware of methods to handle observation of nonterminals only (e.g., that the last two words form a prepositional phrase). We seek, therefore, a mechanism that would admit observational evidence of any form as part of a query about a PCFG, without requiring us to enumerate all consistent parse trees.

We first require a scheme to specify such events as the appearance of symbols at designated points in the parse tree. We can use the indices i and j to delimit the leaf nodes of the subtree, as in the standard chart parsing algorithms. For example, the pp node in the parse tree of Fig. 3.3 is the root of the subtree whose leaf nodes are like and ants, so $i = 3$ and $j = 2$.

However, we cannot uniquely specify a node with these two indices alone. In the branch of the parse tree passing through np, n, and flies, all three nodes have $i = 2$ and $j = 1$. To differentiate them, we introduce the k index, defined recursively. If a node has no child with the same i and j indices, then it has $k = 1$. Otherwise, its k index is one more than the k index of its child. Thus, the flies node has $k = 1$, the noun node above it has $k = 2$, and its parent np has $k = 3$. We have labeled each node in the parse tree of Fig. 3.3 with its (i, j, k) indices.

We can think of the k index of a node as its level of abstraction, with higher values indicating more abstract symbols. For instance, the flies symbol is a specialization of the noun

concept, which, in turn, is a specialization of the np concept. Each possible specialization corresponds to an *abstraction production* of the form $E \rightarrow E'$, that is, with only one symbol on the right-hand side. In a parse tree involving such a production, the nodes for E and E' have identical i and j values, but the k value for E is one more than that of E'. We denote the set of abstraction productions as $P_A \subseteq P$.

All other productions are *decomposition productions*, in the set $P_D = P \setminus P_A$, and have two or more symbols on the right-hand side. If a node E is expanded by a decomposition production, the sum of the j values for its children will equal its own j value, since the length of the original subsequence derived from E must equal the total lengths of the subsequences of its children. In addition, since each child must derive a string of nonzero length, no child has the same j index as E, which must then have $k = 1$. Therefore, abstraction productions connect nodes whose indices match in the i and j components, while decomposition productions connect nodes whose indices differ.

3.2 Bayesian Networks for PCFGs

A *Bayesian network* [36, 33, 25] is a directed acyclic graph where nodes represent random variables, and associated with each node is a specification of the distribution of its variable conditioned on its predecessors in the graph. Such a network defines a joint probability distribution—the probability of an assignment to the random variables is given by the product of the probabilities of each node conditioned on the values of its predecessors according to the assignment. Edges not included in the graph indicate conditional independence; specifically, each node is conditionally independent of its nondescendants given its immediate predecessors. Algorithms for inference in Bayesian networks exploit this independence to simplify the calculation of arbitrary conditional probability expressions involving the random variables.

By expressing a PCFG in terms of suitable random variables structured as a Bayesian network, we could in principle support a broader class of inferences than the standard PCFG algorithms. As we demonstrate below, by expressing the distribution of parse trees for a given probabilistic grammar, we can incorporate partial observations of a sentence as well as other forms of evidence, and determine the resulting probabilities of various features of the parse trees.

3.2.1 PCFG Random Variables

We base our Bayesian-network encoding of PCFGs on the scheme for indexing parse trees presented in Section 3.1.2. The random variable N_{ijk} denotes the symbol in the parse tree at the position indicated by the (i, j, k) indices. Looking back at the example parse tree of Fig. 3.3, a symbol E labeled (i, j, k) indicates that $N_{ijk} = E$. Index combinations not appearing in the tree correspond to N variables taking on the null value nil.

Assignments to the variables N_{ijk} are sufficient to describe a parse tree. However, if we construct a Bayesian network using only these variables, the dependency structure would be quite complicated. For example, in the example PCFG, the fact that N_{213} has the value np would influence whether N_{321} takes on the value pp, even given that N_{141} (their parent in the parse tree) is vp. Thus, we would need an additional link between N_{213} and N_{321}, and, in fact, between all possible sibling nodes whose parents have multiple expansions.

To simplify the dependency structure, we introduce random variables P_{ijk} to represent the productions that expand the corresponding symbols N_{ijk}. For instance, we add the node P_{141}, which would take on the value vp→verb np pp in the example. N_{213} and N_{321} are conditionally independent given P_{141}, so no link between siblings is necessary in this case.

However, even if we know the production P_{ijk}, the corresponding children in the parse tree may not be conditionally independent. For instance, in the chart of Fig. 3.2, entry (1,4) has two separate probability values for the production $S \rightarrow$np vp, each corresponding to different subsequence lengths for the symbols on the right-hand side. Given only the production used, there are again multiple possibilities for the connected N variables: $N_{113} =$ np and $N_{231} =$ vp, or $N_{121} =$ np and $N_{321} =$ vp. All four of these sibling nodes are conditionally dependent since knowing any one determines the values of the other three. Therefore, we dictate that each variable P_{ijk} take on different values for each breakdown of the right-hand symbols' subsequence lengths.

The domain of each P_{ijk} variable therefore consists of productions, augmented with the j and k indices of each of the symbols on the right-hand side. In the previous example, the domain of P_{141} would require two possible values, $S \rightarrow$ np$[1, 3]$vp$[3, 1]$ and $S \rightarrow$ np$[2, 1]$vp$[2, 1]$, where the numbers in brackets correspond to the j and k values, respectively, of the associated symbol. If we know that P_{141} is the former, then $N_{113} =$ np and $N_{231} =$ vp with probability one. This deterministic relationship renders the child N variables conditionally independent of each other given P_{ijk}. We describe the exact nature of this relationship in Section 3.2.3.

Having identified the random variables and their domains, we complete the definition of the Bayesian network by specifying the conditional probability tables representing their interdependencies. The tables for the N variables represent their deterministic relationship with the parent P variables. However, we also need the conditional probability of each P variable given the value of the corresponding N variable, that is, $\Pr(P_{ijk} = E \rightarrow E_1[j_1, k_1] \cdots E_m[j_m, k_m] | N_{ijk} = E)$. The PCFG specifies the relative probabilities of different productions for each nonterminal, but we must compute the probability, $\beta(E, j, k)$ (analogous to the inside probability [7]), that each symbol E_t on the right-hand side is the root node of a subtree, at abstraction level k_t, with a terminal subsequence length j_t.

3.2.2 Calculating β

Algorithm

We can calculate the values for β with a modified version of the dynamic programming algorithm sketched in Section 3.1.1. As in the standard chart-based PCFG algorithms, we can define this function recursively and use dynamic programming to compute its values. Since terminal symbols always appear as leaves of the parse tree, we have, for any terminal symbol $x \in \Sigma$, $\beta(x, 1, 1) = 1$, and for any $j > 1$ or $k > 1$, $\beta(x, j, k) = 0$. For any nonterminal symbol $E \in N$, $\beta(E, 1, 1) = 0$, since nonterminals can never be leaf nodes. For $j > 1$ or $k > 1$, $\beta(E, j, k)$ is the sum, over all productions expanding E, of the probability of that production expanding E and producing a subtree constrained by the parameters j and k.

For $k > 1$, only abstraction productions are possible. For an abstraction production $E \rightarrow E'$, we need the probabilities that E is expanded into E' and that E' derives a string of length j from the abstraction level immediately below E. The former is given by the probability associated with the production, while the latter is simply $\beta(E', j, k - 1)$. According to the independence assumptions of the PCFG model, the expansion of E' is independent of its derivation, so the joint probability is simply the product. We can compute these probabilities for every abstraction production expanding E. Since the different expansions are mutually exclusive events, the value for $\beta(E, j, k)$ is merely the sum of all the separate probabilities.

We assume that there are no abstraction cycles in the grammar. That is, there is no sequence of productions $E_1 \rightarrow E_2$, $\ldots$,$E_{t-1} \rightarrow E_t$, $E_t \rightarrow E_1$, since if such a cycle existed, the above recursive calculation would never halt. The same assumption is necessary for

termination of the standard parsing algorithm. The assumption does restrict the classes of grammars for which such algorithms are applicable, but it will not be restrictive in domains where we interpret productions as specializations, since cycles would render an abstraction hierarchy impossible.

For $k = 1$, only decomposition productions are possible. For a decomposition production $E \rightarrow E_1 E_2 \cdots E_m$ (p), we need the probability that E is thus expanded and that each E_t derives a subsequence of appropriate length. Again, the former is given by p, and the latter can be computed from values of the β function. We must consider every possible subsequence length j_t for each E_t such that $\sum_{t=1}^{m} j_t = j$. In addition, the E_t could appear at any level of abstraction k_t, so we must consider all possible values for a given subsequence length. We can obtain the joint probability of any combination of $\{(j_t, k_t)\}_{t=1}^{m}$ values by computing $\prod_{t=1}^{m} \beta(E_t, j_t, k_t)$, since the derivation from each E_t is independent of the others. The sum of these joint probabilities over all possible $\{(j_t, k_t)\}_{t=1}^{m}$ yields the probability of the expansion specified by the production's right-hand side. The product of the resulting probability and p yields the probability of that particular expansion, since the two events are independent. Again, we can sum over all relevant decomposition productions to find the value of $\beta(E, j, 1)$.

The algorithm in Fig. 3.4 takes advantage of the division between abstraction and decomposition productions to compute the values $\beta(E, j, k)$ for strings bounded by *length*. The array *kmax* keeps track of the depth of the abstraction hierarchy for each subsequence length.

Example Calculations

To illustrate the computation of β values, consider the result of using Charniak's grammar from Fig. 3.1 as its input. We initialize the entries for $j = 1$ and $k = 1$ to have probability one for each terminal symbol, as in Fig. 3.5. To fill in the entries for $j = 1$ and $k = 2$, we look at all of the abstraction productions. The symbols noun, verb, and prep can all be expanded into one or more terminal symbols, which have nonzero β values at $k = 1$. We enter these three nonterminals at $k = 2$, with β values equal to the sum, over all relevant abstraction productions, of the product of the probability of the given production and the value for the right-hand symbol at $k = 1$. For instance, we compute the value for noun by adding the product of the probability of noun→swat and the value for swat, that of noun→flies and flies, and that of noun→ants and ants. This yields the value one,

```
COMPUTE-BETA(grammar,length)
  for each symbol x ∈TERMINALS(grammar)
    β[x,1,1] ← 1.0
  for j ← 1 to length
    kmax[j] ← 0
    for each symbol E ∈NONTERMINALS(grammar)
      β[E,j,1] ← 0.0
    if j > 1
      then
        /* Decomposition phase */
        for each production
            E → E_1 ··· E_m (p) ∈DECOMP-PRODS(grammar)
          for each sequence {j_t}_{t=1}^m such that Σ_t j_t=j
            for each sequence {k_t}_{t=1}^m such that 1 ≤ k_t ≤kmax[j_t]
              result← p
              for t ← 1 to m
                result ← result·β[E_t, j_t, k_t]
              β[E,j,1] ← β[E,j,1]+result
    /* Abstraction Phase */
    while β[E', j, kmax[j] + 1] > 0 for some E'
      kmax[j] ← kmax[j] + 1
      for each production E → E' (p) ∈ABSTRACT-PRODS(grammar)
        if β[E', j, kmax[j]] > 0
          then β[E, j, kmax[j] + 1] ← β[E, j, kmax[j] + 1] + p · β[E', j, kmax[j]]
  return β, kmax
```

Figure 3.4: Algorithm for computing β values.

k	E	$\beta(E,4,k)$	k	E	$\beta(E,3,k)$	k	E	$\beta(E,2,k)$	k	E	$\beta(E,1,k)$
2	S	0.02016	2	S	0.0208	2	S	0.024	4	S	0.06
1	S	0.0832	1	S	0.0576	1	S	0.096	3	np	0.4
	np	0.0672		np	0.176		np	0.08		vp	0.3
	vp	0.1008		vp	0.104		vp	0.12	2	prep	1.0
	pp	0.176		pp	0.08		pp	0.4		verb	1.0
										noun	1.0
									1	like	1.0
										swat	1.0
										flies	1.0
										ants	1.0

Figure 3.5: Final table for sample grammar.

since a noun will always derive a string of length one, at a single level abstraction above the terminal string, given this grammar. The abstraction phase continues until we find S at $k = 4$, for which there are no further abstractions, so we go on to $j = 2$ and begin the decomposition phase.

To illustrate the decomposition phase, consider the value for $\beta(S, 3, 1)$. There is only one possible decomposition production, s$\rightarrow$np vp. However, we must consider two separate cases: when the noun phrase covers two symbols and the verb phrase one, and when the noun phrase covers one and the verb phrase two. At a subsequence length of two, both np and vp have nonzero probability only at the bottom level of abstraction, while at a length of one, only at the third. So to compute the probability of the first subsequence length combination, we multiply the probability of the production by $\beta(\mathsf{np}, 2, 1)$ and $\beta(\mathsf{vp}, 1, 3)$. The probability of the second combination is a similar product, and the sum of the two values provides the value to enter for S.

The other abstractions and decompositions proceed along similar lines, with additional summation required when multiple productions or multiple levels of abstraction are possible. The final table is shown in Fig. 3.5, which lists only the nonzero values.

Complexity

For analysis of the complexity of computing the β values for a given PCFG, it is useful to define d to be the maximum length of possible chains of abstraction productions (i.e., the maximum k value), and m to be the maximum production length (number of symbols on the right-hand side). A single run through the abstraction phase requires time $O(|P_A|)$, and for

each subsequence length, there are $O(d)$ runs. For a specific value of j, the decomposition phase requires time $O(|P_D|j^{m-1}d^m)$, since, for each decomposition production, we must consider all possible combinations of subsequence lengths and levels of abstractions for each symbol on the right-hand side. Therefore, the whole algorithm would take time $O(n[d|P_A| + |P_D|n^{m-1}d^m]) = O(|P|n^m d^m)$.

3.2.3 Network Generation Phase

We can use the β function calculated as described above to compute the domains of random variables N_{ijk} and P_{ijk} and the required conditional probabilities.

Specification of Random Variables

The procedure CREATE-NETWORK, described in Fig. 3.6, begins at the top of the abstraction hierarchy for strings of length n starting at position 1. The root symbol variable, $N_{1n(kmax[n])}$, can be either the start symbol, indicating the parse tree begins here, or nil*, indicating that the parse tree begins below. We must allow the parse tree to start at any j and k where $\beta(S, j, k) > 0$, because these can all possibly derive strings (of any length bounded by n) within the language.

CREATE-NETWORK then proceeds downward through the N_{ijk} random variables and specifies the domain of their corresponding production variables, P_{ijk}. Each such production variable takes on values from the set of possible expansions for the possible nonterminal symbols in the domain of N_{ijk}. If $k > 1$, only abstraction productions are possible, so the procedure ABSTRACTION-PHASE, described in Fig. 3.7, inserts all possible expansions and draws links from P_{ijk} to the random variable $N_{ij(k-1)}$, which takes on the value of the right-hand side symbol. If $k = 1$, the procedure DECOMPOSITION-PHASE, described in Fig. 3.8, performs the analogous task for decomposition productions, except that it must also consider all possible length breakdowns and abstraction levels for the symbols on the right-hand side.

CREATE-NETWORK calls the procedure START-TREE, described in Fig. 3.9, to handle the possible expansions of nil*: either nil* $\rightarrow S$, indicating that the tree starts immediately below, or nil* $\rightarrow$ nil*, indicating that the tree starts further below. START-TREE uses the procedure START-PROB, described in Fig. 3.10, to determine the probability of the parse tree starting anywhere below the current point of expansion.

When we insert a possible value into the domain of a production node, we add it as a

```
CREATE-NETWORK(grammar,length,β,kmax)
  if β[S, length, kmax[length]] > 0.0
    then INSERT-STATE(N_{1(length)kmax[length]},S)
  if START-PROB(β,kmax,length,kmax[length]−1)> 0.0
    then INSERT-STATE(N_{1(length)kmax[length]},nil*)
  for j ← length down-to 1
    for k ← kmax[j] down-to 1
      for i ← 1 to length−j + 1
        for each symbol E ∈DOMAIN(N_{ijk})
          if E ∈NONTERMINALS(grammar)
            then if k > 1
              then ABSTRACTION-PHASE(grammar,E,i,j,k)
              else DECOMPOSITION-PHASE(grammar,E,i,j,k)
            else START-TREE(β,i,j,k)
```

Figure 3.6: Procedure for generating the network.

```
ABSTRACTION-PHASE(grammar,E,i,j,k)
  for each production E → E′ (p) ∈ABSTRACT-PRODS(grammar)
    INSERT-STATE(P_{ijk},E → E′[j, k − 1] (p))
    ADD-PARENT(N_{ij(k−1)},P_{ijk})
    INSERT-STATE(N_{ij(k−1)},E′)
```

Figure 3.7: Procedure for finding all possible abstraction productions.

```
DECOMPOSITION-PHASE(grammar,E,i,j,k)
  for each production E → E_1 E_2 ··· E_m (p) ∈DECOMP-PRODS(grammar)
    for each sequence {j_t}_{t=1}^m such that Σ_t j_t = j
      for each sequence {k_t}_{t=1}^m such that 1 ≤ k_t ≤kmax[j_t]
        if p · Π_t β[E_t, j_t, k_t] > 0
          then INSERT-STATE(P_ijk,E → E_1[j_1, k_1] ··· E_m[j_m, k_m] (p))
            start← i
            for t ← 1 to m
              ADD-PARENT(N_(start)j_t k_t,P_ijk)
              INSERT-STATE(N_(start)j_t k_t,E_t)
              start← start + j_t
```

Figure 3.8: Procedure for finding all possible decomposition productions.

```
START-TREE(β,i,j,k)
  if k > 1
    then if β[S, j, k − 1] > 0.0
      then INSERT-STATE(P_ijk,nil* → S[j, k − 1])
        ADD-PARENT(N_ij(k−1),P_ijk)
        INSERT-STATE(N_ij(k−1),S)
    else if β[S, j − 1, kmax[j − 1]] > 0.0
      then INSERT-STATE(P_ijk,nil* → S[j − 1, kmax[j − 1])
        ADD-PARENT(N_i(j−1)kmax[j−1],P_ijk)
        INSERT-STATE(N_i(j−1)kmax[j−1],S)
  if START-PROB(β,kmax,j,k)> 0.0
    then INSERT-STATE(P_ijk,nil* →nil*)
      if k > 1
        then ADD-PARENT(N_ij(k−1),P_ijk)
          INSERT-STATE(N_ij(k−1),nil*)
        else ADD-PARENT(N_i(j−1)kmax[j−1],P_ijk)
          INSERT-STATE(N_i(j−1)kmax[j−1],nil*)
```

Figure 3.9: Procedure for handling start of parse tree at next level.

START-PROB(β,$kmax$,j,k)

if j=0

then return 0.0

else if k=0

then return START-PROB(β,$kmax$,$j-1$,$kmax[j-1]$)

else return $\beta[S,j,k]$+START-PROB(β,$kmax$,j,$k-1$)

Figure 3.10: Procedure for computing the probability of the start of the tree occurring for a particular string length and abstraction level.

parent of each of the nodes corresponding to a symbol on the right-hand side. We also insert each symbol from the right-hand side into the domain of the corresponding symbol variable. The algorithm descriptions assume the existence of procedures INSERT-STATE and ADD-PARENT. The procedure INSERT-STATE(*node*,*label*) inserts a new state with name *label* into the domain of variable *node*. The procedure ADD-PARENT(*child*,*parent*) draws a link from node *parent* to node *child*.

Specification of Conditional Probability Tables

After CREATE-NETWORK has specified the domains of all of the random variables, we can specify the conditional probability tables. We introduce the lexicographic order $\prec$ over the set $\{(j,k)|1 \leq j \leq n,\ 1 \leq k \leq kmax[j]\}$, where if $j_1 < j_2$ then $(j_1,k_1) \prec (j_2,k_2)$ and if $k_1 < k_2$ then $(j,k_1) \prec (j,k_2)$. For the purposes of simplicity, we do not specify an exact value for each probability $\Pr(X = x|Y)$, but instead specify a weight, $\Pr(X = x_t|Y) \propto \alpha_t$. We compute the exact probabilities through normalization, where we divide each weight by the sum $\sum_t \alpha_t$. The prior probability table for the top node, which has no parents, can be defined as follows:

$$\begin{aligned} \Pr(N_{1n(kmax[n])} = S) &\propto \beta(S,n,kmax[n]) \\ \Pr(N_{1n(kmax[n])} = \mathsf{nil}^*) &\propto \sum_{(j,k)\prec(n,kmax[n])} \beta(S,j,k). \end{aligned}$$

For a given state ρ in the domain of any P_{ijk} node, where ρ represents a production and corresponding assignment of j and k values to the symbols on the right-hand side, of the

form $E \rightarrow E_1[j_1, k_1] \cdots E_m[j_m, k_m]$ (p), we can define the conditional probability of that state as:

$$\Pr(P_{ijk} = \rho | N_{ijk} = E) \quad \propto \quad p \prod_{t=1}^{m} \beta[E_t, j_t, k_t].$$

For any symbol $E' \neq E$ in the domain of N_{ijk}, $\Pr(P_{ijk} = \rho | N_{ijk} = E') = 0$. For the productions for starting or delaying the tree, the probabilities are:

$$\begin{aligned} \Pr(P_{1jk} = \mathsf{nil}^* \rightarrow S[j', k'] | N_{ijk} = \mathsf{nil}^*) \quad &\propto \quad \beta[S, j', k'] \\ \Pr(P_{1jk} = \mathsf{nil}^* \rightarrow \mathsf{nil}^* | N_{ijk} = \mathsf{nil}^*) \quad &\propto \quad \sum_{(j',k') \prec (j,k)} \beta[S, j', k']. \end{aligned}$$

The probability tables for the N_{ijk} nodes are much simpler, since once the productions are specified, the symbols are completely determined. Therefore, the entries are either one or zero. For example, consider the nodes $N_{i_\ell j_\ell k_\ell}$ with the parent node $P_{i'j'k'}$ (among others). For the rule ρ representing $E \rightarrow E_1[j_1, k_1] \cdots E_m[j_m, k_m]$, $\Pr(N_{i_\ell j_\ell k_\ell} = E_\ell | P_{i'j'k'} = \rho, \ldots) = 1$ when $i_\ell = i' + \sum_{t=1}^{\ell-1} j_t$, $j = j_\ell$. For all symbols other than E_ℓ in the domain of $N_{i_\ell j_\ell k_\ell}$, this conditional probability is zero. We can fill in this entry for all configurations of the other parent nodes (represented by the ellipsis in the condition part of the probability), though we know that any conflicting configurations (i.e., two productions both trying to specify the symbol $N_{i_\ell j_\ell k_\ell}$) are impossible. Any configuration of the parent nodes that does not specify a certain symbol indicates that the $N_{i_\ell j_\ell k_\ell}$ node takes on the value nil with probability one.

Network Generation Example

As an illustration, consider the execution of this algorithm using the β values from Fig. 3.5. We start with the root variable N_{142}. The start symbol S has a β value greater than zero here, as well as at points below, so the domain must include both S and nil^*. To obtain $\Pr(N_{142} = S)$, we simply divide $\beta(S, 4, 2)$ by the sum of all β values for S, yielding 0.055728.

The domain of P_{142} is partially specified by the abstraction phase for the symbol S in the domain of N_{142}. There is only one relevant production, $S \rightarrow \mathsf{vp}$, which is a possible expansion since $\beta(\mathsf{vp}, 4, 1) > 0$. Therefore, we insert the production into the domain of P_{142}, with conditional probability one given that $N_{142} = S$, since there are no other possible expansions. We also draw a link from P_{142} to N_{141}, whose domain now includes vp with conditional probability one given that $P_{142} = S \rightarrow \mathsf{vp}$.

To complete the specification of P_{142}, we must consider the possible start of the tree, since the domain of N_{142} includes nil*. The conditional probability of $P_{142} = \mathsf{nil}^* \rightarrow S$ is 0.24356, the ratio of $\beta(S,4,1)$ and the sum of $\beta(S,j,k)$ for $(j,k) \preceq (4,1)$. The link from P_{142} to N_{141} has already been made during the abstraction phase, but we must also insert S and nil* into the domain of N_{141}, each with conditional probability one given the appropriate value of P_{142}.

We then proceed to N_{141}, which is at the bottom level of abstraction, so we must perform a decomposition phase. For the production $S \rightarrow \mathsf{np}\ \mathsf{vp}$, there are three possible combinations of subsequence lengths which add to the total length of four. If np derives a string of length one and vp a string of length three, then the only possible levels of abstraction for each are three and one, respectively, since all others will have zero β values. Therefore, we insert the production s→np[1,3] vp[3,1] into the domain of P_{141}, where the numbers in brackets correspond to the subsequence length and level of abstraction, respectively. The conditional probability of this value, given that $N_{141} = S$, is the product of the probability of the production, $\beta(\mathsf{np},1,3)$, and $\beta(\mathsf{vp},3,1)$, normalized over the probabilities of all possible expansions.

We then draw links from P_{141} to N_{113} and N_{231}, into whose domains we insert np and vp, respectively. The i values are obtained by noting that the subsequence for np begins at the same point as the original string while that for vp begins at a point shifted by the length of the subsequence for np. Each occurs with probability one, given that the value of P_{141} is the appropriate production. Similar actions are taken for the other possible subsequence length combinations. The operations for the other random variables are performed in a similar fashion, leading to the network structure shown in Fig. 3.11.

Complexity of Network Generation

The resulting network has $O(n^2 d)$ nodes. The domain of each N_{i11} variable has $O(|\Sigma|)$ states to represent the possible terminal symbols, while all other N_{ijk} variables have $O(|N|)$ possible states. There are n variables of the former, and $O(n^2 d)$ of the latter. For $k > 1$, the P_{ijk} variables (of which there are $O(n^2 d)$) have a domain of $O(|P_A|)$ states. For P_{ij1} variables, there are states for each possible decomposition production, for each possible combination of subsequence lengths, and for each possible level of abstraction of the symbols on the right-hand side. Therefore, the P_{ij1} variables (of which there are $O(n^2)$) have a domain of $O(|P_D| j^{m-1} d^m)$ states, where we have again defined d to be the maximum value

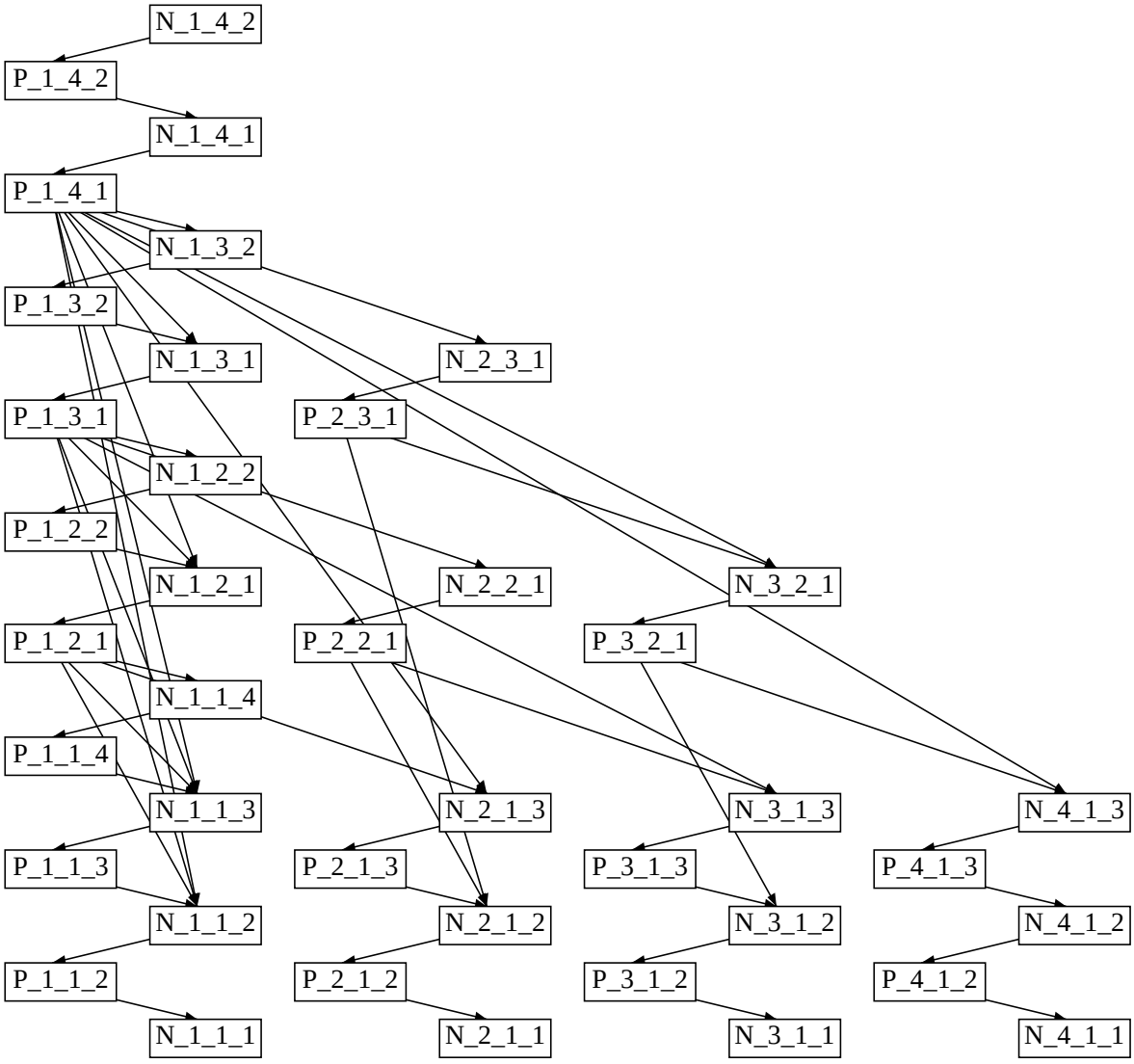

Figure 3.11: Network from example grammar at maximum length 4.

of k, and m to be the maximum production length.

Unfortunately, even though each particular P variable has only the corresponding N variable as its parent, a given N variable could have potentially $O(n)$ P variables as parents. The size of the conditional probability table for a node is exponential in the number of parents, although given that each N can be determined by at most one P (i.e., no interactions are possible), we can specify the table in a linear number of parameters.

If we define T to be the maximum number of entries of any conditional probability table in the network, then the abstraction phase of the algorithm requires time $O(|P_A|T)$, while the decomposition phase requires time $O(|P_D|n^{m-1}d^mT^m)$. Handling the start of the parse tree and the potential space holders requires time $O(T)$. The total time complexity of the algorithm is then $O(n^2|P_D|n^{m-1}d^mT^m+ndT+n^2d|P_A|T+n^2dT) = O(|P|n^{m+1}d^mT^m)$, which dwarfs the time complexity of the dynamic programming algorithm for the β function. However, this network is created only once for a particular grammar and length bound.

3.2.4 PCFG Queries

We can use the Bayesian network to compute any joint probability that we can express in terms of the N and P random variables included in the network. The standard Bayesian network algorithms [36, 33, 13] can return joint probabilities of the form $\Pr(X_{i_1j_1k_1} = x_1, \ldots, X_{i_mj_mk_m} = x_m)$ or conditional probabilities of the form $\Pr(X_{ijk} = x | X_{i_1j_1k_1} = x_1, \ldots, X_{i_mj_mk_m} = x_m)$, where each X is either N or P. Obviously, if we are interested only in whether a symbol E appeared at a particular i, j, k location in the parse tree, we need only examine the marginal probability distribution of the corresponding N variable. Thus, a single network query will yield the probability $\Pr(N_{ijk} = E)$.

The results of the network query are implicitly conditional on the event that the length of the terminal string does not exceed n. We can obtain the joint probability by multiplying the result by the probability that a string in the language has a length not exceeding n. For any j, the probability that we expand the start symbol S into a terminal string of length j is $\sum_{k=1}^{kmax[j]} \beta(S, j, k)$, which we can then sum for $1 \leq j \leq n$. To obtain the appropriate unconditional probability for any query, all network queries reported in this section must be multiplied by $\sum_{j=1}^{n} \sum_{k=1}^{kmax[j]} \beta(S, j, k)$.

Probability of Conjunctive Events

The Bayesian network also supports the computation of joint probabilities analogous to those computed by the standard PCFG algorithms. For instance, the probability of a particular terminal string such as Swat flies like ants corresponds to the probability $\Pr(N_{111} = \mathsf{swat}, N_{211} = \mathsf{flies}, N_{311} = \mathsf{like}, N_{411} = \mathsf{ants})$. The probability of an initial subsequence like Swat flies..., as computed by the LRI algorithm [24], corresponds to the probability $\Pr(N_{111} = \mathsf{swat}, N_{211} = \mathsf{like})$. Since the Bayesian network represents the distribution over strings of bounded length, we can find initial subsequence probabilities only over completions of length bounded by $n - L$.

However, although in this case our Bayesian network approach requires some modification to answer the same query as the standard PCFG algorithm, it needs no modification to handle more complex types of evidence. The chart parsing and LRI algorithms require complete sequences as input, so any gaps or other uncertainty about particular symbols would require direct modification of the dynamic programming algorithms to compute the desired probabilities. The Bayesian network, on the other hand, supports the computation of the probability of any evidence, regardless of its structure. For instance, if we have a sentence Swat flies ... ants where we do not know the third word, a single network query will provide the conditional probability of possible completions $\Pr(N_{311}|N_{111} = \mathsf{swat}, N_{211} = \mathsf{flies}, N_{411} = \mathsf{ants})$, as well as the probability of the specified evidence $\Pr(N_{111} = \mathsf{swat}, N_{211} = \mathsf{flies}, N_{411} = \mathsf{ants})$.

This approach can handle multiple gaps, as well as partial information. For example, if we again do not know the exact identity of the third word in the sentence Swat flies ... ants, but we do know that it is either swat or like, we can use the Bayesian network to fully exploit this partial information by augmenting our query to specify that any domain values for N_{311} other than swat or like have zero probability. Although these types of queries are rare in natural language, domains like speech recognition and plan recognition often require this ability to reason when presented with noisy observations.

We can answer queries about nonterminal symbols as well. For instance, if we have the sentence Swat flies like ants, we can query the network to obtain the conditional probability that like ants is a prepositional phrase, $\Pr(N_{321} = \mathsf{pp}|N_{111} = \mathsf{swat}, N_{211} = \mathsf{like}, N_{311} = \mathsf{like}, N_{411} = \mathsf{ants})$. We can answer queries where we specify evidence about nonterminals within the parse tree. For instance, if we know that like ants is a prepositional phrase, the input to the network query will specify that $N_{321} = \mathsf{pp}$, as well as specifying the terminal

symbols.

Alternate network algorithms can compute the most probable state of the random variables given the evidence, instead of a conditional probability [36, 10, 13]. For example, consider the case of possible four-word sentences beginning with the phrase Swat flies.... The probability maximization network algorithms can determine that the most probable state of terminal symbol variables N_{311} and N_{411} is like flies, given that N_{111} =swat, N_{211} =flies, and N_{511} =nil.

Probability of Disjunctive Events

We can also compute the probability of disjunctive events through multiple network queries. If we can express an event as the union of mutually exclusive events, each of the form $X_{i_1 j_1 k_1} = x_1 \wedge \cdots \wedge X_{i_m j_m k_m} = x_m$, then we can query the network to compute the probability of each, and sum the results to obtain the probability of the union. For instance, if we want to compute the probability that the sentence Swat flies like ants contains any prepositions, we would query the network for the probabilities $\Pr(N_{i12} = \mathsf{prep} | N_{111} = \mathsf{swat}, N_{211} = \mathsf{like}, N_{311} = \mathsf{like}, N_{411} = \mathsf{ants})$, for $1 \leq i \leq 4$. In a domain like plan recognition, such a query could correspond to the probability that an agent performed some complex action within a specified time span.

In this example, the individual events are already mutually exclusive, so we can sum the results to produce the overall probability. In general, we ensure mutual exclusivity of the individual events by computing the conditional probability of the conjunction of the original query event and the negation of those events summed previously. For our example, the overall probability would be $\Pr(N_{112} = \mathsf{prep} | \mathcal{E}) + \Pr(N_{212} = \mathsf{prep}, N_{112} \neq \mathsf{prep} | \mathcal{E}) + \Pr(N_{112} = \mathsf{prep}, N_{112} \neq \mathsf{prep}, N_{212} \neq \mathsf{prep} | \mathcal{E}) + \Pr(N_{112} = \mathsf{prep}, N_{112} \neq \mathsf{prep}, N_{212} \neq \mathsf{prep}, N_{312} \neq \mathsf{prep} | \mathcal{E})$, where $\mathcal{E}$ corresponds to the event that the sentence is Swat flies like ants.

The Bayesian network provides a unified framework that supports the computation of all of the probabilities described here. We can compute the probability of any event $\mathcal{E}$, where $\mathcal{E}$ is a set of mutually exclusive events $\{X_{i_{t1} j_{t1} k_{t1}} \in \mathcal{X}_{t_1} \wedge \cdots \wedge X_{i_{tm_t} j_{tm_t} k_{tm_t}} \in \mathcal{X}_{t_{m_t}}\}_{t=1}^{h}$ with each X being either N or P. We can also compute probabilities of events where we specify relative likelihoods instead of strict subset restrictions. In addition, given any such event, we can determine the most probable configuration of the uninstantiated random variables. Instead of designing a new algorithm for each such query, we have only to express the query

in terms of the network's random variables, and use any Bayesian network algorithm to compute the desired result.

Complexity of Network Queries

Unfortunately, the time required by the standard network algorithms in answering these queries is potentially exponential in the maximum string length n, though the exact complexity will depend on the connectedness of the network and the particular network algorithm chosen. The algorithm in our current implementation uses a great deal of preprocessing in compiling the networks, in the hope of reducing the complexity of answering queries. Such an algorithm can exploit the regularities of our networks (e.g., the conditional probability tables of each N_{ijk} consist of only zeroes and ones) to provide reasonable response time in answering queries. Unfortunately, such compilation can itself be prohibitive and will often produce networks of exponential size. There exist Bayesian network algorithms [14, 12] that offer greater flexibility in compilation, possibly allowing us to limit the size of the resulting networks, while still providing acceptable query response times.

Determining the optimal tradeoff will require future research, as will determining the class of domains where our Bayesian network approach is preferable to existing PCFG algorithms. It is clear that the standard dynamic programming algorithms are more efficient for the PCFG queries they address. For domains requiring more general queries of the types described here, the flexibility of the Bayesian network approach may justify the greater complexity. In such domains, the alternative would be to design and implement algorithms aimed at each specific class of query. The Bayesian network approach, on the other hand, provides a unified platform that supports all classes of queries (within the limits of the specified random variables) without any necessary re-design.

3.3 PCFGs for Plan Recognition

Plan recognition domains may need the generality of queries supported by the Bayesian network approach. A PCFG representation of a plan recognition domain would define a correspondence between symbols in the grammar and plan events in the problem domain. The PCFG represents the same probability distribution over parse trees that the problem domain exhibits over plan instantiations. Then, when we observe plan events, we use the corresponding grammatical evidence as input to a PCFG inference algorithm to compute

Drive	→	Stay Drive	(0.8)	Drive	→	Exit	(0.005)
Drive	→	Left Drive	(0.05)	2-Left	→	Left Left	(1.0)
Drive	→	Right Drive	(0.05)	2-Right	→	Right Right	(1.0)
Drive	→	2-Left Drive	(0.01)	Pass	→	Right Left	(0.1)
Drive	→	2-Right Drive	(0.01)	Pass	→	Left Right	(0.9)
Drive	→	Pass Drive	(0.075)				

Figure 3.12: A probabilistic context-free grammar representing a simplified traffic model.

the posterior probabilities necessary to answer the recognizing agent's queries.

In our simplified traffic domain, our observations consist of the possible lane changes a driver can make, represented by the terminal symbols Left, Right, and Stay, where the last indicates that the driver has stayed in its current lane. The nonterminal symbols represent the more complex actions, formed from combinations of these low-level lane changes. The PCFG of Figure 3.12 represents one possible driver model, starting from the top-level symbol Drive.

According to this PCFG, a driver's plan execution consists of a sequence of episodes, each rooted by a node labeled Drive and terminating in a sequence of low-level lane change actions. The parse tree of Figure 3.13 represents one possible sequence, of probability 2.8125×10^{-7}, where the driver executes two passing maneuvers before exiting. Since this probability distribution over parse trees corresponds to a distribution over possible plan instantiations, we can use the PCFG query algorithms to handle the analogous plan recognition queries. For instance, the PCFG chart parsing algorithm can compute the probability of any observed action sequence, as well as the most probable plan instantiation explaining that sequence.

However, in plan recognition, the recognizing agent usually interacts with the observed agent before it finishes its execution. For instance, in the traffic example, plan recognition is useless once planning is complete, since every plan instantiation terminates with the observed car having exited the highway, at which point no interactions will take place with a recognizing car still on the highway. Plan recognition has need of many other possible queries left unaddressed by the standard PCFG algorithms. As stated in Section 3.1.2, the standard algorithms cannot handle partial strings, but a plan recognizer rarely has complete observations of another agent's past actions. In addition, when agents can communicate, a recognizing agent may gain information about subplans, corresponding to evidence of

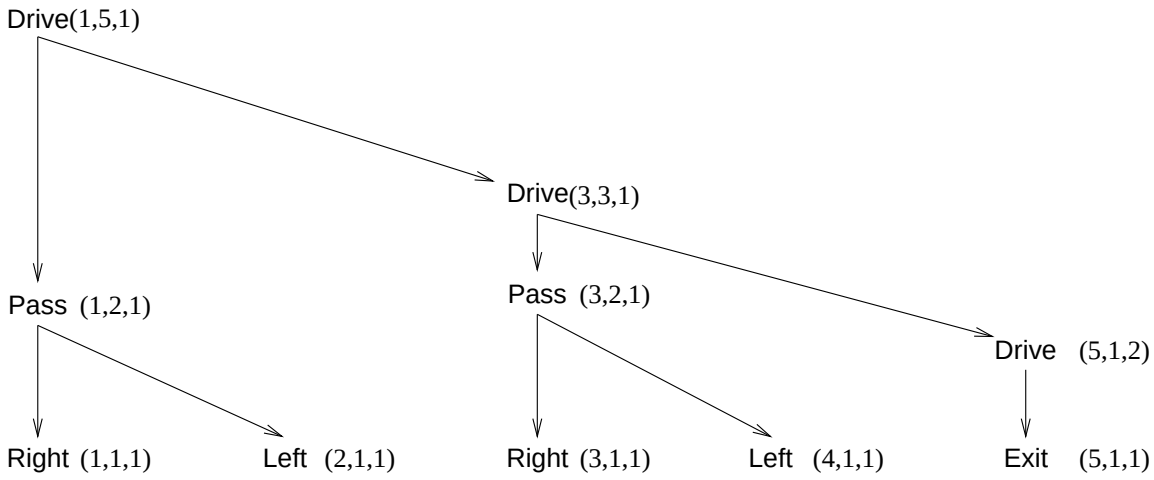

Figure 3.13: Sample parse tree from traffic PCFG with (i, j, k) indices labeled.

nonterminals in the parse tree, that cannot be exploited by the standard algorithms.

Therefore, the generality of the Bayesian network approach to PCFG inference is of great advantage in plan recognition. We can use the algorithms of Sections 3.2.3 and 3.2.2 to generate a Bayesian network representation of possible plan instantiations. The PCFG length bound now represents a bound on the length of the action sequence, which may correspond to temporal durations as well exceeding a fixed bound on the length of the with a fixed length of time. We can then use the algorithms of Section 3.2.4 to answer recognition queries. For instance, if we had observed only the first passing maneuver of the parse tree in Figure 3.13, we might query the network for the probability that the agent stays in its current lane in the next period of time, i.e. $\Pr(N_{311} = \mathsf{Stay} | N_{111} = \mathsf{Right}, N_{211} = \mathsf{Left})$. Perhaps more importantly, we can answer queries about possible subplans in the next period of time ($\Pr(N_{321} = \mathsf{Pass} | N_{111} = \mathsf{Right}, N_{211} = \mathsf{Left})$). The Bayesian network's ability to incorporate evidence about any available random variables allows us to answer queries when we have evidence about subplans. For instance, if the recognizing agent knows that the observed driver has decided to pass, we can use the network to compute the relative likelihoods of passing on the left or right ($\Pr(N_{311} = \mathsf{Left} | N_{111} = \mathsf{Right}, N_{211} = \mathsf{Left}, N_{321} = \mathsf{Pass})$). In general, we can compute any conditional probability of any subplans or actions occurring at a given time within the plan instantiation, given all observed subplans and actions.

CHAPTER 4

Context Sensitivity

Although the Bayesian network algorithms generalize the set of answerable queries, the independence assumptions of the PCFG model remain overly restrictive for many domains. By definition, the probability of applying a particular PCFG production to expand a given nonterminal is independent of what symbols have come before and of what expansions are to occur after. In the parse tree of Figure 3.13, the probability that the driver passes on the right in the second pass is fixed at 0.1, even though we would reasonably expect a higher likelihood given that the first pass was also on the right. A more obvious problem arises because the example PCFG does not keep track of the driver's current lane. Presumably, there is a limit to the number of left lane changes a driver can perform in succession before reaching the boundary of the highway, but we cannot represent that restriction within the production structure of the sample PCFG.

Of course, we may be able to correct the model by expanding the set of nonterminals to encode contextual information, and thus preserving the structure of the PCFG model. For instance, we could define the grammatical symbols to be all possible combinations of subplans/actions and lane positions. We would then have expansions of the form $\langle \mathsf{Pass}, \mathsf{leftlane} \rangle \rightarrow \langle \mathsf{Right}, \mathsf{leftlane} \rangle \ \langle \mathsf{Left}, \mathsf{middlelane} \rangle$ (1.0). The symbol on the right-hand side would indicate that the observed driver is executing a passing maneuver from the leftmost lane by first performing a right lane change from the leftmost lane and then a left lane change from the middle lane, where the lane context changes as a result of the right lane change first executed. This production has probability one because the driver cannot pass on the left from the leftmost lane.

However, such adjustments to the grammar can obviously lead to an unsatisfactory increase in complexity for both the design and use of the model. Instead, we could use an

alternate model that relaxes the PCFG independence assumptions. Such a model would need a more complex production and/or probability structure to allow complete specification of the distribution, as well as modified inference algorithms for manipulating this distribution.

4.1 Direct Extensions to Network Structure

The Bayesian network representation of the probability distribution provides a possible basis for exploring such extensions to an underlying PCFG model. The networks generated by the algorithms of this paper implicitly encode the PCFG assumptions through assignment of a single nonterminal node as the parent of each production node. This single link indicates that the expansion is conditionally independent of all other nondescendant nodes, once we know the value of this nonterminal. We could extend the context-sensitivity of these expansions within our network formalism by adding links among these production nodes.

We could introduce some context sensitivity even without adding any links. Since each production node has its own conditional probability table, we can define the production probabilities to be a function of the (i, j, k) index values. For instance, we can impose limits on the duration of a subplan's expansion by varying the production probability appearing over different string lengths, as encoded by the j index. In such cases, we can modify the standard PCFG representation so that the probability information associated with each production is a function of i, j, and k, instead of a constant. The dynamic programming algorithm of Fig. 3.4 can be easily modified to handle production probabilities that depend on j and k. However, a dependency on the i index as well would require making all three indices parameters of β and introducing an additional loop to the existing algorithm to cover all possible i values. Then, we would have to replace any reference to the production probability, in either the dynamic programming or network generation algorithm, with the appropriate function of i, j, and k.

Alternatively, we may introduce additional dependencies on other nodes in the network. A PCFG extension that conditions the production probabilities on the parent of the left-hand side symbol has already proved useful in modeling natural language [8]. For instance, we could extend the simple traffic PCFG from Figure 3.12 so that Left and Right are nonterminal symbols expanded further into accelerations, as well as lateral movements (e.g. Right $\rightarrow$ Move-Right Decelerate (0.6)). In such a grammar, a choice between accelerating or

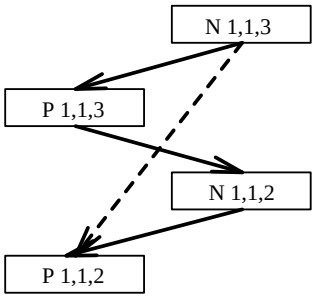

Figure 4.1: Subnetwork incorporating parent symbol dependency.

decelerating as an expansion of a right lane change would depend on whether the lane change were part of a more complicated passing maneuver or not. In this case, each production has a set of associated probabilities, one for each nonterminal symbol that is a possible parent of the symbol on the left-hand side. This new probability structure requires modifications to both the dynamic programming and the network generation algorithms. We must first extend the probability information of the β function to include the parent nonterminal as an additional parameter. It is then straightforward to alter the dynamic programming algorithm of Fig. 3.4 to correctly compute the probabilities in a bottom-up fashion.

The modifications for the network generation algorithm are more complicated. Whenever we add P_{ijk} as a parent for some symbol node $N_{\hat{i}\hat{j}\hat{k}}$, we also have to add N_{ijk} as a parent of $P_{\hat{i}\hat{j}\hat{k}}$. For example, the dotted arrow in the subnetwork of Fig. 4.1 represents the additional dependency of P_{112} on N_{113}. We must add this link because N_{112} is a possible child nonterminal, as indicated by the link from P_{113}. The conditional probability tables for each P node must now specify probabilities given the current nonterminal and the parent nonterminal symbols. We can compute these by combining the modified β values with the conditional production probabilities.

Returning to the example from the beginning of this section, we may want to condition the production probabilities on expansions already chosen. As a first approximation to such context sensitivity, we can imagine a model where each production has an associated set of probabilities, one for each terminal symbol in the language. Each represents the conditional probability of the particular expansion given that the corresponding terminal symbol occurs immediately previous to the subsequence derived from the nonterminal symbol on the left-hand side. For instance, a driver content to stay in its current lane in the previous time period is more likely to be so again in the next time period. Therefore, we may wish to condition the probability of the production Drive$\rightarrow$Stay Drive on the event that the previous

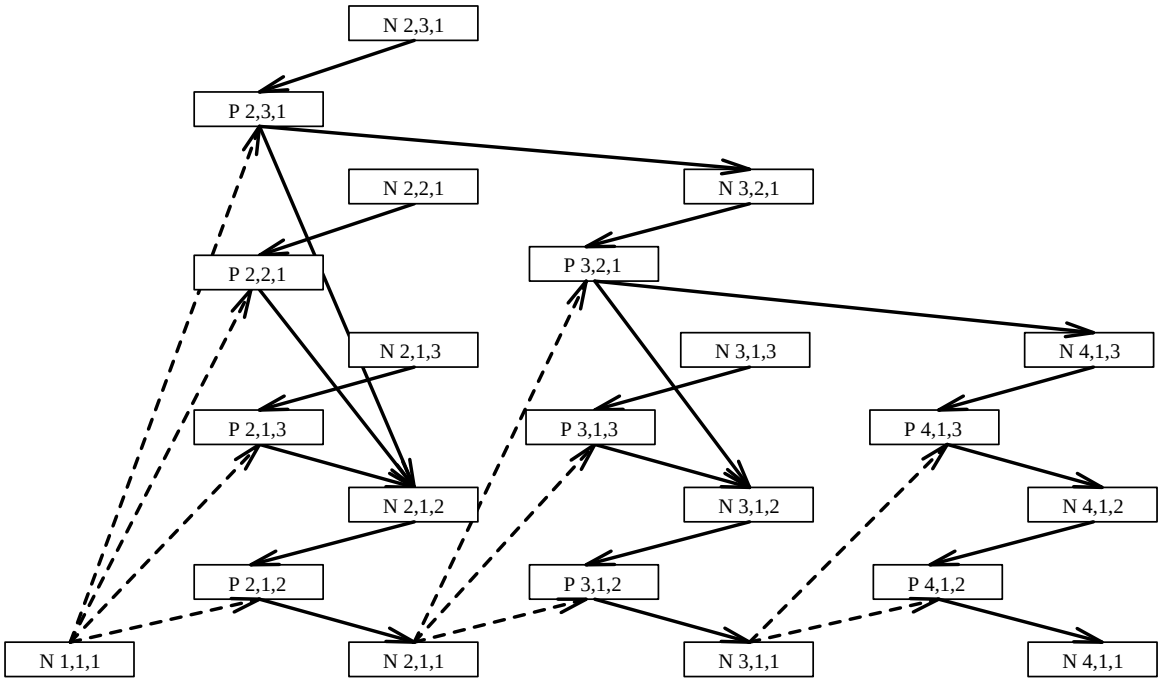

Figure 4.2: Subnetwork capturing dependency on previous terminal symbol.

action chosen was Stay.

Again, our β function requires an additional parameter, and we need a modified version of the dynamic programming algorithm to compute its values. However, the network generation algorithm needs to introduce only one additional link from N_{i11} for each $P_{(i+1)jk}$ node. The dashed arrows in the subnetwork of Fig. 4.2 reflect the additional dependencies introduced by this context sensitivity, using the network example from Fig. 3.11. The P_{1jk} nodes are a special case, with no preceding terminal, so the steps from the original algorithm are sufficient.

We can extend this conditioning to cover preceding terminal *sequences* rather than individual symbols. For instance, when only three lanes are available, we can prevent a driver from performing three consecutive left lane changes by setting the probability of the production Drive→Left Drive to be zero when the preceding terminal sequence is two consecutive Lefts. In general, each production could have an associated set of probabilities, one for each possible terminal sequence of length bounded by some parameter h. The β function now requires an additional parameter specifying the preceding sequence. The network generation algorithms must then add links to P_{ijk} from nodes $N_{(i-h)11},\ldots,N_{(i-1)11}$, if $i \geq h$, or from $N_{111},\ldots,N_{(i-1)11}$, if $i < h$. The conditional probability tables then specify

the probability of a particular expansion given the symbol on the left-hand side and the preceding terminal sequence.

These suggested extensions are merely patchwork modifications, addressing very specific forms of context sensitivity. In general, the Bayesian networks currently generated contain a set of random variables sufficient for expressing arbitrary parse tree events, so we can introduce context sensitivity by adding the appropriate links to the production nodes from the events on which we wish to condition expansion probabilities. We can thus account for the dependency among a driver's choice between passing on the left or right by introducing links between the production nodes that could potentially represent those choices. Once we have the correct network, we can use any of the query algorithms from Section 3.2.4 to produce the corresponding conditional probability.

In many cases, we may wish to account for external influences, such as explicit context representation in natural language problems or influences of the current world state in planning. For instance, if we are processing multiple sentences, we may want to draw links from the symbol nodes of one sentence to the production nodes of another, to reflect thematic connections. In the traffic example, we can introduce random variables representing the positions and speeds of other cars along the highway and draw links from these new variables to the production nodes representing the observed driver's plan choices. As long as our network can include random variables to represent the external context, then we can represent the dependency by adding links from the corresponding nodes to the appropriate production nodes and altering the conditional probability tables to reflect the effect of the context.

4.2 Modifications to the Grammatical Model

Context sensitivities expressed as incremental changes to the network dependency structure represent only a local relaxation of the conditional independence assumptions of the PCFG model. More global models of context sensitivity will require a radically different grammatical form and probabilistic interpretation framework. However, the few incremental changes already discussed suggest a wide variety of possible new grammatical formalisms, each with different degrees of expressive power and inferential complexity.

Ignoring the stochastic component of the model for the moment, *context-sensitive* grammars (CSGs) [19] can handle many of the dependencies present in plan recognition domains.

In a CSG, every production has an associated context representing an intermediate sequence of symbols that must be present for the specified expansion to be possible. More precisely, productions take the form $\xi_L X \xi_R \rightarrow \xi_L \xi_M \xi_R$, with $\xi_\cdot \in (N \cup \Sigma)^*$. Such a production states that we can expand the symbol X into the sequence ξ_M at any point when X occurs with the string ξ_L immediately to its left and ξ_R immediately to its right. Many of the suggested model extensions fit very easily into this production format. For instance, a production of the form Stay Drive→Stay Stay Drive would represent an observed car's staying in the current lane conditioned on the fact that it stayed in that lane during the previous time period.

The generality of context-sensitive productions complicates possible probabilistic extensions. In the PCFG model, we could virtually guarantee a coherent probability distribution over terminal strings by making the one restriction that productions over a particular nonterminal symbol have a total probability of one. We can add likelihood information to context-sensitive productions, but we cannot determine, a priori, which expansions may be applicable at a given point in the parse (due to the different possible contexts), so a set of fixed production values may not produce weights that sum to one in a particular context. We instead must normalize the likelihoods from the set of productions that are currently applicable at a given intermediate stage of the parse tree and use the corresponding normalized likelihood as the probability of choosing a particular production at that point. The set of applicable productions must consider all nonterminal symbols left unexpanded, because the relaxation of the context-free assumption prevents the modular probability assignment of the PCFG model.

For instance, consider the parse tree of Figure 3.13 as being generated by a probabilistic context-sensitive-grammar (PCSG). At the stage of the parse where we have selected only the production Drive→Pass Drive, we have an intermediate string of Pass Drive. At this point, we must consider all context-free productions of the form Pass→ ξ and Drive→ ξ, as well as the context-sensitive productions of the form Pass Drive→ ξ Drive and Pass Drive→Pass ξ. The weight of each production, normalized in some manner, would be the production probability. Suppose that we choose the production Pass→Right Left, producing the intermediate string Right Left Drive. At this point, there is only one nonterminal symbol left unexpanded, but there are three possible contexts: Drive, Left Drive, and Right Left Drive. We again perform the normalization over the set of corresponding weights to obtain the production probabilities.

As in the PCFG case, this PCSG model assumes that each subsequent random production selection is independent of the previous and future choices. Therefore, the probability of a particular derivation sequence is uniquely determined as the product of all of the individual production choices, although the result will be sensitive to the order in which we apply the productions. We could then define a probability distribution over all strings in the context-sensitive language so that the probability of a particular string is the sum of the probabilities over all possible derivation sequences for that string. However, specifying the production weights to model a given distribution is complicated. In the simple traffic parse tree of the example, we considered the productions of the form $\mathsf{Drive} \rightarrow \xi$ at both intermediate stages, but weighed against different sets of candidates. In general, it is difficult to choose a fixed weight for a given production and still guarantee the desired probability that the production will be the chosen expansion of its nonterminal symbol.

Performing inference with this PCSG model is even less straightforward. We could potentially modify the PCFG algorithms to create a Bayesian network with the proper conditional dependency structure to represent a PCSG distribution. However, a PCSG distribution is sensitive to the order of production application, so we lose the modularity necessary for the chart-based representation and the dynamic programming algorithms. More limiting is the increased complexity of the dependency structure. Even if we find a compact set of random variables, a correct Bayesian network requires links to each production node from all potential context nodes, and, with full context sensitivity, all symbols that are not direct ancestors or descendants are potential context symbols. In fact, unless we can restrict the form of the context sensitivity, the Bayesian network must represent, at least implicitly, the joint space over all possible intermediate strings. Without the decomposition into separate symbols allowable under PCFG independence assumptions, the Bayesian network approach will be impractical for all but the simplest problems.

4.3 State Dependency in Grammatical Model

The impracticality of this PCSG model indicates that a more restricted grammatical model is necessary. Looking back at the special needs of plan recognition as outlined in Figure 2.1 (PR model from UAI paper), we see that the agent's choice of plan depends on its mental state. In fact, many of the discussed examples of context sensitivity are, in reality, examples of a common dependency on the agent's mental state. For instance, a recognizing

agent, having observed a pass on the right, now has a different belief about the type of the driver being observed, specifically that the driver is of a type more likely to pass on the right than the average driver. Thus, the recognizing agent will have a higher degree of belief in the second pass taking place on the right because of this updated belief about the driver's type, not because of any direct relationship between the two passing episodes. Likewise, we can model the proposed dependency of the probability of a production like Drive→Stay Drive on the previous terminal symbol's being Stay by again introducing a common parent variable in the observed agent's mental state, representing its satisfaction with its current lane. Most planning languages are capable of modeling such context dependency, where context here refers to the planning context (mental state), not grammatical context (plan/action symbols).

Within a PCFG model of a planning agent, the probability of applying a particular production must be independent of the current state of the planning process, given the subplan being expanded. Therefore, as discussed briefly in the opening of this chapter, the only way to represent the production probability's dependency on the agent's mental state is to expand the space of nonterminal symbols to represent combinations of subplans and mental states. The large space of possible mental states produces an even larger space of nonterminal symbols. Modeling problem domains is much more difficult in this joint space. More ominously, the complexity of inference grows exponentially with each additional feature of an agent's mental state.

CHAPTER 5

Probabilistic State-Dependent Grammars

As an alternative, we can create an explicit model of the agent's mental state by introducing a separate random variable with limited interactions with the underlying PCFG. The resulting *probabilistic state-dependent grammar* (PSDG) is still capable of representing the dependency of plan choices on mental state, as well as that of the effects on context and plan choices, while still imposing enough structure to simplify inference and domain specification. Section 5.1 defines the components of a PSDG and the independence assumptions needed to coherently define a probability distribution over its language.

5.1 Specification of PSDG Language Model

A probabilistic state-dependent grammar is a tuple $\langle \Sigma, N, S, Q, P, \pi_0, \pi_1 \rangle$, where Σ, N, and S are the same as in a PCFG, Q^t is a time-indexed random variable representing a state space (beyond the grammatical symbols) with domain Q, and π_0 and π_1 specify the probability distribution governing the process that generates the states. The productions P take the same form as in a PCFG, $E \rightarrow \xi\ (p)$, except that now, p is a function of the state, $Q \rightarrow [0,1]$. Each such production states that the conditional probability of expanding E into the sequence ξ, given that the current state $Q^t = q$, is $\Pr(q)$. The relevant current state to consider when expanding a symbol whose terminal substring starts in position i of the overall terminal string (index value i as defined in Section 3.1.2) is Q^{i-1}. For each nonterminal symbol $E \in N$, consider all rules of the form $E \rightarrow \xi_\ell\ (p_\ell)$. We require that $\sum_\ell p_\ell(q) = 1$ for all points $q \in Q$.

The function π_0 specifies the distribution over the initial values of the state variable Q, i.e. $\Pr(Q^0 = q) = \pi_0(q)$. The function π_1 specifies the distribution over subsequent values

$$
\begin{array}{llll}
0) & \text{Drive} \rightarrow \text{Stay} & (p_0(q) = \cdots) \\
1) & \text{Drive} \rightarrow \text{Left} & \left(p_1(q) = \begin{cases} 0 & \text{if Lane}(q) = \text{left-lane} \\ \cdots & \end{cases}\right) \\
2) & \text{Drive} \rightarrow \text{Right} & (p_2(q)) \\
3) & \text{Drive} \rightarrow \text{Pass} & (p_3(q)) \\
4) & \text{Drive} \rightarrow \text{Exit} & (p_4(q)) \\
5) & \text{Pass} \rightarrow \text{Left Right} & (p_5(q)) \\
6) & \text{Pass} \rightarrow \text{Right Left} & (p_6(q))
\end{array}
$$

Figure 5.1: A PSDG representation of a simplified traffic domain.

of Q. The value of Q^t is conditionally independent of past values of Q given the value of Q^{t-1} and the terminal symbol chosen in the interval between $t-1$ and t:

$$
\begin{aligned}
\Pr(Q^t = q_t | Q^0 = q_0, \ldots, Q^{t-1} = q_{t-1}, N_{t11} = x) &= \Pr(Q^t = q_t | Q^{t-1} = q_{t-1}, N_{t11} = x) \\
&= \pi_1(q_{t-1}, x, q_t)
\end{aligned}
$$

This Markov property of the state variables allows us to ignore state values before time $t-1$ when computing the probability distribution over Q^t, once we have observed Q^{t-1} and the terminal symbol at position t. The value of the state at time t is also independent of all symbols in the parse tree with $i \leq t$ (other than N_{t11}), when we know the value of the state and terminal symbol at time t.

We can often simplify the definition of the production probability functions p and the state distribution functions π_0 and π_1 by viewing the state as a conjunction of somewhat orthogonal features representing invidual aspects of the environment, as well as the agent's beliefs, preferences, and capabilities. Thus, production probabilities are functions of only those features that influence the choice in expanding a particular symbol. Likewise, the distribution over a particular feature can depend on other certain features, without having to depend on all. Most of this chapter refers to the state as a single variable for intelligibility, but the algorithms for network generation and inference exploit factored state representations as well.

As an illustration of the interaction between the grammar and the state variable, consider the PSDG of Figure 5.1, presenting the set of productions from a simplified model of driving plans. The start symbol, Drive, represents a single decision-making episode from driving along a three-lane highway. The terminal symbols, Stay, Left, Right, and Exit, represent actions with the obvious effects on the driving lane. The intermediate symbol Pass represents the driver's ability to pass another car by first changing over to the next lane (to either the

left or right) and then returning to the original lane once beyond the car being passed.

Figure 5.1 partially specifies the production probability functions based on a state space Q representing all possible combinations of the positions and speeds of all cars within the observed driver's view, as well as the state of the observed driver's own car and its preferences. In general, we need a separate representation of the observed agent's beliefs about the state of the world to correctly model the decision process, if these beliefs can deviate from the recognizing agent's own beliefs, implicit in the model of the world dynamics and subsequent observations. To simplify this illustration, we assume that the driver has perfect observations, so we represent only the true state of the world and take the agent's beliefs to be equivalent. The state also includes information about the agent's preferences about driving speed, distance from other cars, intended exit, etc. Each component of the state has a corresponding function that returns that component's value of a given point in the state space, e.g. the function $\mathsf{Lane}(q)$ returns the lane position of the observed car in state q, whether left-lane, middle-lane, or right-lane. Figure 5.2 illustrates a possible parse tree for a plan generated from this PSDG. The picture labeled Q^0 in the bottom left corner of the diagram represents the initial state. The solid black rectangle is the driver whose planning process we are trying to recognize. The white rectangles are the other cars on the highway that the driver of interest must consider when planning.

The driver's plan originates with the start symbol Drive and must choose among the five possible expansions shown in Figure 5.1. We compute the likelihood of each production by applying the corresponding probability function p_ℓ to the current state Q^0, since the substring rooted at Drive begins at the start of the overall terminal sequence (i.e., the i index is 1). In the sample parse tree of Figure 5.2, the driver has selected the production $\mathsf{Drive} \rightarrow \mathsf{Pass}$, whose probability is $p_3(Q^0)$. The symbol Pass also has an i index of 1, so we compute $p_5(Q^0)$ and $p_6(Q^0)$ to determine the probability of passing on the left versus passing on the right. In the example, the driver has chosen to pass on the left, so it first executes a Left action.

This branch of the parse tree has now reached a leaf node, so we can apply the state transition probability, $\pi_1(Q^0, \mathsf{Left}, Q^1)$, to compute a distribution over possible values of Q^1. The diagram shows one possible value where the driver has moved into the leftmost lane (as a result of selecting the Left action) and moved beyond the other two cars. The state value Q^1 forms the context when expanding any symbols with $i = 2$. In this example, only the terminal symbol Right has $i = 2$, but in general, we must consider production probability

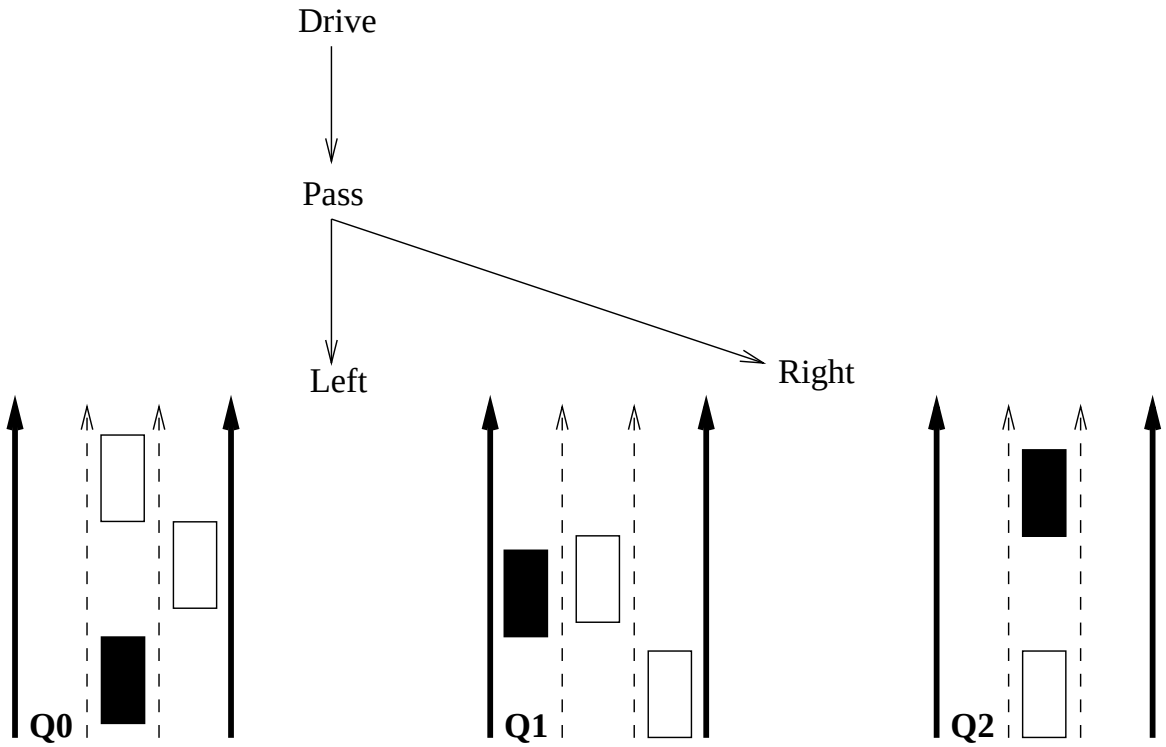

Figure 5.2: Sample PSDG parse tree from the traffic domain.

functions of the form $\Pr(Q^1)$ in expanding the branch until we reach a leaf node. Once we have the leaf node, we again apply the state transition probability, $\pi_1(Q^1, \mathsf{Right}, Q^2)$ in this case.

5.2 Inference on PSDGs

The benefit of the PSDG representation extends beyond simplifying domain specification. Although we can perform inference on a given PSDG by generating the corresponding PCFG and then using the algorithms of Section 3.2.4, the explosion in the size of the symbol space could lead to prohibitive costs for answering queries. Fortunately, we can exploit the independence assumptions in the PSDG model for more efficient inference.

5.2.1 Generation of Bayesian Networks for PSDGs

The Bayesian network model used in Section 3.2.4 already contains a set of random variables sufficient for representing the production choices of the PSDG. For a network generated for a length bound of n, we can introduce $n + 1$ nodes representing the state variables Q^0, Q^1,..., Q^n. The node Q^0 has no parents, with π_0 specifying its probability distribution. Every state node Q^i for $i > 0$ has the Q^{i-1} and N_{i11} nodes as parents, with π_1 specifying the conditional probability table.

Each production node P_{ijk} needs an additional link from node Q^{i-1} because the production probabilities now depend on the value of that state node. To specify the conditional probability tables of these production nodes, we also need to compute the β function, that is, the probability that a given nonterminal symbol derives a subtree with particular j and k indices, given its parent symbol. Because of the link from Q^{i-1}, the β function must now consider all state values that are potential contexts for the productions at that point in the parse tree. In addition, we must consider how the context changes for each of the symbols on the right-hand side from the expansion of its previous siblings.

For instance, consider a production node, P_{121}, whose domain contains the two expansions of Pass from the PSDG of Figure 5.1. In this case, there is only one possible breakdown of substring lengths, i.e. both Left and Right have substring length 1. The probability of the expansion $\mathsf{Pass} \rightarrow \mathsf{Left\ Right}$, given that N_{121} =Pass and $Q^0 = q_0$, is the product of $p_5(q_0)$ (production probability), $\beta(\mathsf{Left}, 1, 1, q_0)$ (probability that Left derives a subtree of length 1 and $k = 1$ given context q_0), and the sum over all states $q_1 \in Q$ of the product

of $\pi_1(q_0, \mathsf{Left}, q_1)$ (probability that q_1 is the resulting context after expanding Left in q_0) and $\beta(\mathsf{Right}, 1, 1, q_1)$ (probability that Right derives a subtree of length 1 and $k = 1$ given context q_1).

Although we can alter the dynamic programming algorithm of Figure 3.4 to include state information, the summation over intermediate states is of much greater concern. In this simple example, we can compute the probability of an intermediate state q_1 by directly applying the π_1 function. However, if the right-hand side of the production included nonterminal symbols, instead of the terminal symbols Left and Right, we would have to compute the transition probabilities indirectly, requiring another dynamic programming algorithm. More importantly, we only had to compute a summation over one intermediate state q_1, but each additional symbol on the right-hand side requires another summation over the entire state space. In fact, the time complexity of the revised β algorithm for a string length of n and for a maximum hierarchy depth of d is $O(|Q|^{m+1}|P|n^m d^m)$, where we assume that the maximum number of symbols on the right-hand side, m, is less than the maximum string length, n. Fortunately, we incur this cost only once, during the generation of the Bayesian network, so in most cases, we can afford the additional complexity.

5.2.2 Dynamic Bayesian Network Representation of PSDGs

However, regardless of the complexity of the network generation algorithms, the resulting Bayesian network represents a probability distribution over sequences only within some length bound. For domains where we do not know the maximum sequence length, we run the risk of having to regenerate a new network on the fly, where the complexity of the generation algorithm may be unacceptable. In addition, even if we know the maximum sequence length and can generate a Bayesian network covering all the cases encountered in the queries, the resulting network may be too large to answer the queries with satisfactory timeliness. These difficulties make it impossible to generate a single, operational network that covers all possible queries in real-world problem domains.

Fortunately, most plan recognition domains do not require the full range of queries covered by these Bayesian networks. Temporal constraints make most queries irrelevant. For instance, the sequence of our observations roughly obeys temporal ordering, so we will not receive new direct evidence about random variables arbitrarily far in the past. It is also unlikely that we will want to compute posterior probabilities over possible subplans and actions beyond some fixed time into the past or future. In other words, we confine our

interest in the agent's planning process to a window of fixed duration.

The query DAG formalism citeDarwiche97 supports the generation of a specialized structures that represent the same distribution as the Bayesian network, but that answer a restricted set of queries more efficiently than possible with the more general network structure. In a plan recognition domain where we know what fixed set of queries the recognizing agent requires, we could generate DAG representation of the PSDG distribution to avoid some of the complexity of the static Bayesian network representation. Unfortunately, the query DAG generation process is similar to the Bayesian network compilation process, which is prohibitively costly for even simple PSDGs. Therefore, it is unlikely that a query DAG approach would significantly alleviate the complexity problem.

Dynamic Bayesian networks (DBNs) [27] make specific restrictions on the set of possible queries. If we are modeling some sequential stochastic process, then a DBN represents a window of a fixed number of random variables, as does our static network representation. However, the DBN also represents the probability distribution between successive phases of the processes evolution, so the window can move to represent future variables as well. We can use existing DBN inference algorithms to enter evidence and answer queries about any of the random variables within the current window. When we move the window, the DBN algorithms maintain an exact representation of the underlying distribution, incorporating all of the evidence received so far, even of variables beyond the window.

For a PCFG or PSDG distribution, we can view the position from left to right as providing the temporal ordering of the underlying process. We can thus view any symbols generated along the same branch (i.e., with the same i index) as belonging to the same time slice. A DBN representation must include a specification of the distribution within a given time slice, so we have to model the interaction of all of the symbol and production variables with the same i index. To allow the DBN inference algorithms to move the window, we must model the dependency of the random variables in one time slice on the variables in the previous slice. Both the intra- and inter-slice specifications take the form of Bayesian networks, so much of our work from Section 5.2.1 carries over to the DBN generation algorithms as well.

Random Variables for DBN Representation of PSDG

With a DBN representation of the distribution over parse trees, we can no longer use the chart-based indexing scheme, because we may not know j, the length of the substring

rooted at a particular node, until some time into the future. However, although the fixed-length window of the DBN prevents the representation of entire subtrees, we can represent entire branches if we include variables to represent an entire path from root node to leaf node. Therefore, we do not use a bottom-up labeling scheme, since we do not know the length of a given path beforehand, but must work top-down instead. The ℓ index represents the distance of a given symbol from the root node, which always is the start symbol of the grammar. The root node has $\ell = 1$, and all other symbol nodes has an ℓ index of one more than their parent nodes. If we view a particular branch as a hierarchy of pending subplans, then the ℓ index indicates the level of the hierarchy of a subplan, with 1 corresponding to the top level plan.

For now, we assume no cycles exist within the productions, allowing us to define d as the largest possible ℓ. We relax this assumption to permit limited recursion in the next subsection. We also index random variables by time t, resulting in a set of symbol variables N_ℓ^t and production variables P_ℓ^t. The time index t has a particular relationship with the i and j indices of the static networks, namely if $N_{ijk} = A$, then there is some ℓ such that $N_\ell^t = A$ for all t such that $i \leq t < i + j$. In other words, while the current time is within the subsequence derived from A, then A appears at the symbol node $\ell - 1$ steps away from the root node. In the example from Figure 5.2, the top-level Drive has $\ell = 1$, as do all root nodes, so N_1^t =Drive for all t within the range of this parse tree. In this example, the terminal string has length 2, so $N_1^1 = N_2^1$ =Drive. Its child node, Pass, has $\ell = 2$, but it too spans the entire execution, so $N_2^1 = N_2^2$ =Pass. Its children have $\ell = 3$ and span only a single time each, so N_3^1 =Left and N_3^2 =Right.

For a PSDG, we must also include a state variable Q^t to represent the sequence of contexts. The probability table for Q^0 takes its values from π_0. Subsequent Q^t nodes have Q^{t-1} as a parent, as well as any N_ℓ^t nodes that have terminal symbols in their domain. As an alternative, we can introduce a new node Σ^t representing the terminal symbol at position t in the final sequence (N_{t11} using the (i, j, k) indices). The state nodes then depend on only this new Σ^t node and the previous state node Q^{t-1}. We can define the conditional probability table for these Q^t nodes using π_1. The Σ^t node depends on all P_ℓ^t nodes that have productions with terminal symbols on their right-hand side, and we remove all terminal symbols from the domains of the N_ℓ^t nodes.

The value of production node P_ℓ^t indicates the production chosen to expand the symbol in N_ℓ^t, as well as what symbol on the right-hand side is being currently expanded as $N_{\ell+1}^t$ or

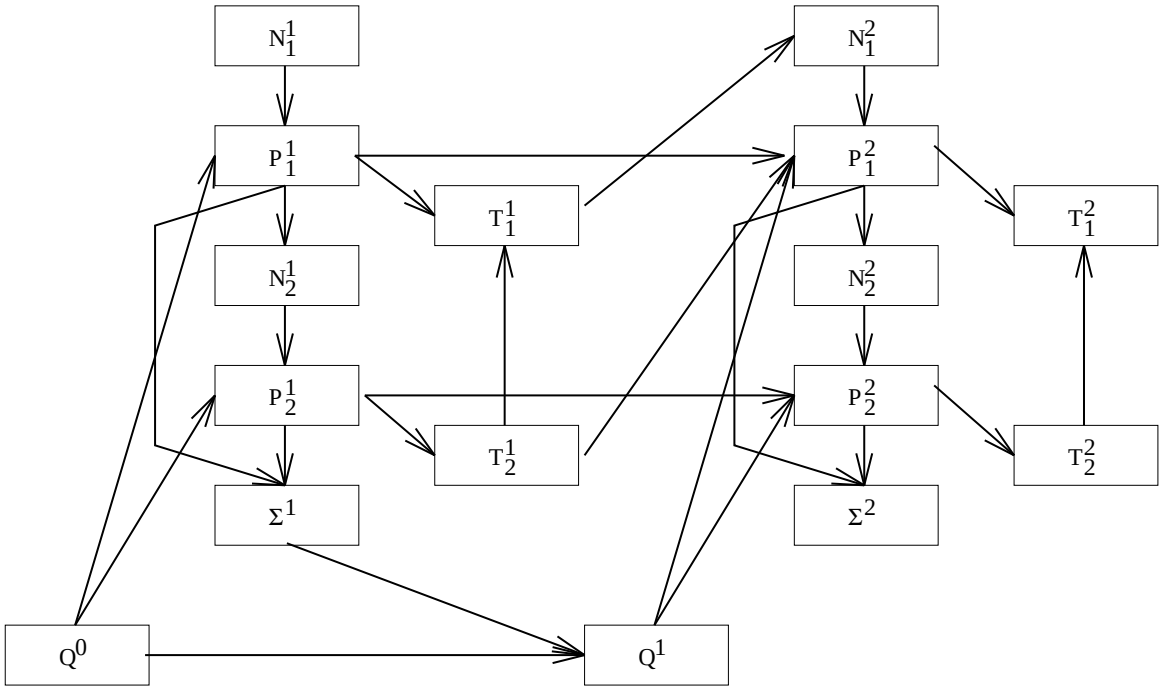

Figure 5.3: DBN representation for PSDG of Figure 5.1 over two time slices.

currently executed as Σ^t. More precisely, if $P_\ell^t = \langle X \rightarrow Y_1 Y_2 \cdots Y_m, b \rangle$, then N_ℓ^t must be X. If the right-hand symbol Y_b (indexed by the position indicator b from the production node's value) is a terminal symbol, then $\Sigma^t = Y_b$; otherwise, the child symbol node $N_{\ell+1} = Y_b$. We can define the position indicator b in terms of the (i, j, k) indices of Y_b: $i \leq t < i + j$. With this position indicator, the child symbol nodes (including Σ^t) take on values deterministically given the value of their parent production node.

For instance, a DBN representation of the simple PSDG of Figure 5.1 would begin with a root node of N_1^1 with a domain of {Drive, nil}. The domain of P_1^1 is {$\langle 1,1\rangle$,$\langle 2,1\rangle$,$\langle 3,1\rangle$,$\langle 4,1\rangle$,$\langle 5,1\rangle$,nil}, where the ordered pair $\langle a, b\rangle$ indicates production a at position b of the right-hand side. Only production 3 produces a nonterminal symbol, leading to a N_2^1 node with a domain of {Pass,nil}. The domain of node P_2^1 is then {$\langle 5,1\rangle$,$\langle 5,2\rangle$,$\langle 6,1\rangle$,$\langle 6,2\rangle$,nil}, allowing us to differentiate between the two components of a passing maneuver. None of the productions in this set produce nonterminal symbols, so there are no more N nodes and $d = 2$. The terminal symbol node Σ^1 has a domain of {Stay,Left,Right,Exit,nil}. The nodes for time 1 appear to the left of the DBN in Figure 5.3.

It is straightforward to generate the conditional probability tables for these nodes from

the PSDG. We compute the probability that P_ℓ^1 takes on the value $\langle a, 1\rangle$ by applying the corresponding p_a function to the values of the parent state node Q^0. The values $\langle 5, 2\rangle$ and $\langle 6, 2\rangle$ have zero probability because we cannot move on to the second stage of the passing maneuver until the first is complete. The root node $N_1^1 =$Drive with probability one. The other symbol nodes, N_2^1 and Σ^1, are completely determined given their parents' values.

The symbol and production variables for time 2 have a more complicated dependency structure because of their relationship to the variables at time 1, although the domains of the variables are unchanged. For instance, $N_1^1 =$Drive if its expansion has not *terminated* after time 1. The expansion of N_ℓ^t terminates if the value of P_ℓ^t is $\langle X \rightarrow Y_1 Y_2 \cdots Y_m, m\rangle$ and if either $Y_m \in \Sigma$ or if the expansion of $N_{\ell+1}^t = Y_m$ terminates. In other words, an expansion of a symbol terminates at time t if there are no productions in an intermediate state below that symbol. In terms of the (i, j, k) indices, the expansion of symbol N_{ijk} terminates at time $i + j - 1$. The expansion of a symbol $N_\ell^t =$nil is terminated by definition.

We can simplify the dependency structure for time 2 by introducing boolean random variables T_ℓ^t that are true iff the expansion of N_ℓ^t terminates. Each T_ℓ^t node has P_ℓ^t and $T_{\ell+1}^t$ as a parent, except for T_d^t, which has only P_d^t as a parent. The conditional probability table of each T_ℓ^t node represents the deterministic relationship of the definition of termination. In our example, T_2^t is false iff P_2^t is $\langle 5, 1\rangle$ or $\langle 6, 1\rangle$, where there is still an additional symbol left to expand. Termination at the level above, T_1^t is false iff P_1^t is $\langle 3, 1\rangle$ and T_2^t is false. We can now define the root node N_1^{t+1} as taking on the value of Drive iff T_1^t is false, indicating that the expansion of Drive is still incomplete.

The production nodes P_ℓ^2 also depend on the values of these termination nodes. For instance, if the expansion of symbol $N_{\ell+1}^t$ does not terminate, then the value of production node P_ℓ^{t+1} does not change from the value of P_ℓ^t because we are still expanding the same symbol on the right-hand side. If the expansion of $N_{\ell+1}^t$ does terminate, then the value of P_ℓ^{t+1} takes the value of P_ℓ^t and move the position indicator to the next symbol on the right-hand side. If P_ℓ^t was already at the last symbol on the right-hand side, then the expansion of N_ℓ^t has terminated and we must choose a new production for P_ℓ^{t+1} based on whatever symbol now appears N_ℓ^{t+1}, where we compute the probabilities as for $t = 1$, by applying the production probability functions to the relevant state, Q^t.

For the example DBN of Figure 5.3, many of the dependencies created by the automatic generation algorithm sketched so far are unnecessary, but they are included here for illustrative purposes. The node P_1^2 depends on its corresponding symbol node N_1^2, the previous

production node P_1^1, and its child's termination node T_2^1. If T_2^1 is false (during a passing maneuver), we simply copy the value of P_1^1 into P_1^2. Otherwise, we would normally move on to the next position on the right-hand side, but in this case, there is only one symbol on the right-hand side of all of the productions in domain of P_1^t, so we have to choose a new production based on N_1^2. However, if the production of P_1^1 has terminated, then the entire parse tree has been completely expanded and $N_1^2 = \mathsf{nil}$, so P_1^2 is nil as well.

The node P_2^2 depends on its symbol node N_2^2 and the previous production node P_2^1. There is no dependency on any termination node because, at the bottom of the hierarchy, all children are terminal symbols, for whom termination would always be true. If P_2^1 is $\langle 5, 1\rangle$ or $\langle 6, 1\rangle$, then at time 2, we move on to the second stage of the expansions, so P_2^2 will be $\langle 5, 2\rangle$ or $\langle 6, 2\rangle$, respectively. For any other values of P_2^1, the parse tree has completed, so N_2^2, and thus P_2^2, will be nil.

Recursion in PSDGs

We no longer have a length bound over sequences in the DBN representation, unlike the static Bayesian network representation. However, we do need to fix the maximum depth, d, of the symbol hierarchy. Otherwise, we would have to generate new DBNs to introduce new symbol and production variables as needed. If the productions introduce any possible cycles, then there is no bound on the length of parse tree branches. Therefore, we cannot allow any recursion in the productions unless we alter our definition of the ℓ index.

Fortunately, if we assume that we do not have to answer queries arbitrarily far into the past, we can allow recursive productions of the form $X \to Y_1 Y_2 \cdots Y_{m-1} X$, where the $Y_b \neq X$. The Y_b children are treated as before, with ℓ being one more than the ℓ value of the X on the left-hand side. However, the X on the right-hand side now has the *same* ℓ value as the X on the left-hand side. Therefore, after the expansion of Y_{m-1} at $N_{\ell+1}^t$ has terminated, we move on to the final symbol, X, which is expanded from N_ℓ^{t+1}. We choose a new production for X at P_ℓ^{t+1}, so we no longer have any record that the X derived from the production $X \to Y_1 Y_2 \cdots Y_{m-1} X$. In addition, because we keep X at the same ℓ value, we have no record of how many levels of nesting have taken place.

As long as we have no need of this lost information, even this limited form of recursion provides a more expressive PSDG language. We can now expand the grammar of Figure 5.1, which can capture only a single episode of a driver's decision-making, to include recursive productions that model a driver performing a sequence of such episodes. Figure 5.4 provides

0)	Drive	$\rightarrow$	Stay Drive	$(p_0(q))$
1)	Drive	$\rightarrow$	Left Drive	$(p_1(q))$
2)	Drive	$\rightarrow$	Right Drive	$(p_2(q))$
3)	Drive	$\rightarrow$	Pass Drive	$(p_3(q))$
4)	Drive	$\rightarrow$	Exit	$(p_4(q))$
5)	Pass	$\rightarrow$	Left Right	$(p_5(q))$
6)	Pass	$\rightarrow$	Right Left	$(p_6(q))$

Figure 5.4: A probabilistic state-dependent grammar with recursive productions.

one possible model that repeats the original decision-making procedure until the driver chooses to exit the highway. The complete DBN generation algorithm is presented in Figures 5.5, 5.6, 5.7, and 5.7.

PSDG Inference Using DBNs

The generation algorithm creates a DBN of as many time slices as specified. The nodes of time 1 represent the initial state of the PSDG process, while the slices for time $t > 1$ have the same dependency on time $t - 1$. Once we have this DBN representation, we can use the standard DBN algorithms for incorporating evidence, computing posterior probabilities, and moving the time window into the future. We have the same generality of queries within the time window as we did within the static Bayesian networks of Section 3.2.4.

Unfortunately, the complexity of DBN inference is likely to be impractical for most PSDGs. Even though the DBN inference algorithm focuses on only the fixed-length time window, it needs to maintain a belief state sufficient to capture all of the evidence incorporated in the past. This belief state obviously cannot explicitly represent all the evidence. Instead, the DBN inference algorithm represents the belief state in terms of the random variables of a single time slice, but with an altered dependency structure that must capture the effects of removing all the nodes from times previous. Nodes that were conditionally independent given these past nodes are dependent now that we no longer explicitly represent their parents.

Thus, the sparsely connected time slices of the original DBN belie the high connectivity of the belief state. In fact, inter-slice connections along each level of the hierarchy and upon the state variable lead to a fully connected belief state in most cases. In other words, the belief state is equivalent to the joint distribution over all possible combinations of state and

GENERATE-DYNAMIC-BELIEF-NET-SLICE1(*grammar,d*)
  for each state variable Q_r^0
    for each value q in the domain of Q_r
      INSERT-STATE(Q_r^0, q)
    for each parent state variable Q_p^0 of Q_r^0
      ADD-PARENT(Q_r^0, Q_p^0)
    for each configuration $\mathbf{q_p}$ of parent state variables of Q_r^0
      for each domain value q in the domain of Q_r
        $\Pr(Q_r^0 = q | \mathbf{Q_p^0} = \mathbf{q_p}) \leftarrow \pi_{0Q_r}(q, \mathbf{q_p})$
  INSERT-STATE(N_1^1, S) ; $\Pr(N_1^1 = S) \leftarrow 1.0$
  INSERT-STATE(N_1^1, nil) ; $\Pr(N_1^1 = \text{nil}) \leftarrow 0.0$
  for $\ell \leftarrow 1$ **to** $d + 1$
    ADD-PARENT($P_{\ell-1}^1, N_{\ell-1}^1$)
    for each symbol $X \in N_{\ell-1}^1$
      for each expansion $a = X \rightarrow Y_1 \cdots Y_m$ $(p) \in$PRODUCTIONS(*grammar*)
        for each state variable Q_r on which p depends
          ADD-PARENT($P_{\ell-1}^1, Q_r^0$) ; INSERT-STATE($P_{\ell-1}^1, \langle a, 1 \rangle$)
          if $Y_1 \in N$
            then INSERT-STATE(N_ℓ^1, Y_1)
            else ADD-PARENT($\Sigma^1, P_{\ell-1}$) ; INSERT-STATE(Σ^1, Y_1)
          for each configuration $\mathbf{q}$ of parent states $\mathbf{Q_r}$
            $\Pr(P_{\ell-1}^1, \langle a, 1 \rangle | N_{\ell-1}^1 = X, \mathbf{Q_r} = \mathbf{q}) \leftarrow \Pr(\mathbf{q})$
    ADD-TERMINATION-NODES(*grammar*, $\ell, d, 1$)
    if $\ell \leq d$ **then** ADD-SYMBOL-NODES(*grammar*, $\ell, 1$)
  for each $x \in \Sigma^1$
    for each $\langle a = X \rightarrow Y_1 \cdots Y_m, b \rangle \in$ the domain of any parent P_ℓ^1 of Σ^1
      if $x = Y_b$
        then $\Pr(\Sigma^1 = x | P_\ell^1 = \langle a, b \rangle) \leftarrow 1.0$
        else $\Pr(\Sigma^1 = x | \text{no } P_\ell^1 = \langle a, b \rangle) \leftarrow 0.0$

Figure 5.5: Pseudocode for algorithm generating the initial variables of a DBN representation of a given PSDG distribution.

GENERATE-DYNAMIC-BELIEF-NET-SLICE2($grammar$,d)
  for each state variable Q_r^1
    for each value q in the domain of Q_r
      INSERT-STATE(Q_r^1, q)
    if transition probability of Q_r depends on terminal symbol
      then ADD-PARENT(Q_r^1, Σ^1)
    for each parent state variable Q_p^t (t being 0 or 1) of Q_r^1
      ADD-PARENT(Q_r^1, Q_p^t)
    for each configuration $\mathbf{q_p}$ of parent state variables of Q_r^1
      for each $x \in \Sigma$
        for each domain value q in the domain of Q_r
          $\Pr(Q_r^1 = q | \mathbf{Q_p^t} = \mathbf{q_p}, \Sigma^1 = x) \leftarrow \pi_{1Q_r}(q_p, x, q)$
  ADD-PARENT(N_1^2, T_1^1) ; INSERT-STATE(N_1^2, S)
  $\Pr(N_1^2 = S | T_1^1) \leftarrow 0.0$; $\Pr(N_1^2 = S | \neg T_1^1) \leftarrow 1.0$
  INSERT-STATE(N_1^2, nil) ; $\Pr(N_1^2 = \mathsf{nil} | T_1^1) \leftarrow 1.0$; $\Pr(N_1^2 = \mathsf{nil} | \neg T_1^1) \leftarrow 0.0$
  for $\ell \leftarrow 1$ **to** $d+1$
    ADD-PARENT($P_{\ell-1}^2, N_{\ell-1}^2$) ; ADD-PARENT($P_{\ell-1}^2, P_{\ell-1}^1$) ; ADD-PARENT($P_{\ell-1}^2, T_\ell^1$)
    for each symbol $X \in N_{\ell-1}^2$
      for each expansion $a = X \rightarrow Y_1 \cdots Y_m$ (p) $\in$PRODUCTIONS($grammar$)
        for each state variable Q_r on which p depends
          ADD-PARENT($P_{\ell-1}^0, Q_r^1$)
          for $b \leftarrow 1$ **to** m
            if $Y_b \neq X$
              INSERT-STATE($P_{\ell-1}^2, \langle a, b \rangle$)
              if $Y_b \in N$
                then INSERT-STATE(N_ℓ^2, Y_b)
                else ADD-PARENT($\Sigma^2, P_{\ell-1}^2$) ; INSERT-STATE(Σ^2, Y_b)
              for each configuration $\mathbf{q}$ of parent states $\mathbf{Q_r}$
                $\Pr(P_{\ell-1}^2, \langle a, b \rangle | N_{\ell-1}^2 = X, T_\ell^1, P_{\ell-1}^1 = \langle X' \rightarrow Y_1' \cdots Y_{m'}', m' \rangle, \mathbf{Q_r} = \mathbf{q})$
                  $\leftarrow \Pr(\mathbf{q})$
                $\Pr(P_{\ell-1}^2, \langle a, b \rangle | N_{\ell-1}^2 = X, T_\ell^1, P_{\ell-1}^1 = \langle a, b-1 \rangle, \mathbf{Q_r} = \mathbf{q}) \leftarrow 1.0$
                $\Pr(P_{\ell-1}^2, \langle a, b \rangle | N_{\ell-1}^2 = X, \neg T_\ell^1, P_{\ell-1}^1 = \langle a, b \rangle, \mathbf{Q_r} = \mathbf{q}) \leftarrow 1.0$
    ADD-TERMINATION-NODES($grammar, \ell, d, 2$)
    if $\ell \leq d$ **then** ADD-SYMBOL-NODES($grammar, \ell, 2$)
  for each $x \in \Sigma^2$
    for each $\langle a = X \rightarrow Y_1 \cdots Y_m, b \rangle \in$ the domain of any parent P_ℓ^2 of Σ^2
      if $x = Y_b$
        then $\Pr(\Sigma^2 = x | P_\ell^2 = \langle a, b \rangle) \leftarrow 1.0$
        else $\Pr(\Sigma^2 = x | \text{no } P_\ell^2 = \langle a, b \rangle) \leftarrow 0.0$

Figure 5.6: Pseudocode for algorithm generating the variables at time 2 of a DBN representation of a given PSDG distribution.

ADD-TERMINATION-NODES($grammar, \ell, d, t$)

  ADD-PARENT(T^t_ℓ, P^t_ℓ)

  if $\ell < d$

    ADD-PARENT($T^t_\ell, T^t_{\ell+1}$)

  for each $\langle a = X \rightarrow Y_1 \cdots Y_m, b \rangle \in P^t_\ell$

    /* Do not execute $\neg T^t_{\ell+1}$ cases if $\ell = d$: */

    if $a = m$

      then $\Pr(T^t_\ell | P_\ell = \langle a, b \rangle, T^t_{\ell+1}) \leftarrow 1.0$

        $\Pr(T^t_\ell | P_\ell = \langle a, b \rangle, \neg T^t_{\ell+1}) \leftarrow 0.0$

      else $\Pr(T^t_\ell | P_\ell = \langle a, b \rangle, T^t_{\ell+1}) \leftarrow 0.0$

        $\Pr(T^t_\ell | P_\ell = \langle a, b \rangle, \neg T^t_{\ell+1}) \leftarrow 0.0$

Figure 5.7: Pseudocode for procedure generating the termination variables for a DBN representation of a given PSDG distribution.

ADD-SYMBOL-NODES($grammar, \ell, t$)

  ADD-PARENT($N^t_\ell, P^t_{\ell-1}$)

  for each $Y \in N^t_\ell$

    for each $\langle a = X \rightarrow Y_1 \cdots Y_m, b \rangle \in P^t_{\ell-1}$

      if $Y = Y_b$

        then $\Pr(N^t_\ell = Y | P^t_{\ell-1} = \langle a, b \rangle) leftarrow 1.0$

        else $\Pr(N^t_\ell = Y | P^t_{\ell-1} = \langle a, b \rangle) \leftarrow 0.0$

Figure 5.8: Pseudocode for procedure generating the nonterminal symbol variables for a DBN representation of a given PSDG distribution.

parse tree branches. This space is far too large to allow belief state specification, let alone evidence propagation.

5.2.3 PSDG Inference through Direct Manipulation of the Belief State

The high connectivity of the DBN belief state arises from its reliance on conditional independence as its structuring property. Fortunately, the PSDG process exhibits weaker forms of independence as well, which we can exploit for specialized inference algorithms. If we re-examine the relationship of the production nodes P_ℓ^t on the previous time slice, we notice that there are two possibilities: either $N_{\ell+1}^{t-1}$ has not terminated, or $N_{\ell+1}^{t-1}$ has terminated but we have not finished the production at $P_\ell^{t-1} = \langle X \rightarrow Y_1 Y_2 \cdots Y_m, b \rangle$, $b < m$. In the former case, $P_\ell^t = P_\ell^{t-1}$, while in the latter, $P_\ell^t = \langle X \rightarrow Y_1 Y_2 \cdots Y_m, b+1 \rangle$. In both cases, the relationship is deterministic. If neither case holds, then we are choosing a new production based on N_ℓ^t and Q^{t-1}, *independent of the symbols and productions of the previous time slice.*

The algorithms presented in this section exploit this particular independence property. The Bayesian network representation of our beliefs over symbols in a given time slice grows too large because there is insufficient conditional independence within the PSDG model. If we use a specialized representation of our beliefs, we can exploit the specific PSDG independence in maintaining a compact summary of our posterior beliefs over symbols in the current time slice, conditioned on all of our past observations. The resulting *belief state* operates similarly to the DBN window in that it contains sufficient probabilistic information to answer plan recognition queries given the current evidence and to create a revised belief state upon receiving new evidence.

Completely Observable States

As a first step, we first analyze the case where we are interested in posterior probabilities given a sequence of observations of the state variable, $\mathcal{E}^t \equiv Q^0, \ldots, Q^t$. The queries of interest are either plan prediction probabilities, $\Pr(P_\ell^t | \mathcal{E}^{t-1})$ or plan explanation probabilities, $\Pr(P_\ell^t | \mathcal{E}^t)$, from which we can easily derive the corresponding symbol probabilities. Figure 5.9 lists the conditional probability tables that form B^t, the belief state for time t, including the probabilities over states, symbols, productions, and termination, conditioned on the evidence received. Inference also requires various other probabilities, e.g., symbol probabilities conditioned on termination and termination probabilities conditioned on par-

Field		Contents
B_Q^t	=	$\Pr(\mathcal{E}^{t-1}\|\mathcal{E}^{t-2})$
$B_N^t(\ell, X)$	=	$\Pr(N_\ell^t = X\|\mathcal{E}^{t-1})$
$B_P^t(\ell, \langle a, b\rangle)$	=	$\Pr(P_\ell^t = \langle a, b\rangle \,\|\mathcal{E}^{t-1})$
$B_\Sigma^t(x)$	=	$\Pr(\Sigma^t = x\|\mathcal{E}^{t-1})$
$B_T^t(\ell)$	=	$\Pr(T_\ell^t\|\mathcal{E}^{t-1})$
$B_{T\|N}^t(\ell, X)$	=	$\Pr(T_\ell^t\|\mathcal{E}^{t-1}, N_\ell^t = X)$

Figure 5.9: Components of belief state for support of PSDG inference.

ticular symbols. The space complexity of the belief state is $O(d|P|m)$, dominated by the distribution over production states, B_P^t.

Belief State Initialization Figure 5.10 provides the pseudocode for the belief state initialization process. The initial belief state begins with $B_Q^1 = \Pr(\mathcal{E}^0)$, easily obtained from the prior probability function π_0. We can then work top down, starting with N_1^1 which takes on the value of the start symbol (Drive in the example from Figure 5.4) with probability one. Then, for each level ℓ of the hierarchy, we compute the probability distribution over the production node P_ℓ^1. The probability that the production node is $\langle a, 1\rangle$, for production a with symbol X_a on the left and with probability function p_a, is the product of the probability that $N_\ell^1 = X_a$ given state Q^0 and the production probability $p_a(Q^0)$. Using the PSDG from Figure 5.4, the probability that $P_1^1 = \langle 0, 1\rangle$ would be $p_0(Q^0)$, while the probability that $P_2^1 = \langle 5, 1\rangle$ would be $\Pr(N_2^1 = \mathsf{Pass}|Q^0)p_5(Q^0)$. At time 1, no production node can have a value of $\langle a, b > 1\rangle$, since we are expanding the first symbol on the right-hand side.

For each production node P_ℓ^1 that has a probability ρ of taking on the value $\langle a, 1\rangle$, we add ρ to the probability that the child symbol is the first symbol on the right-hand side. If this symbol X is a nonterminal, then we add ρ to the probability that $N_{\ell+1}^1 = X$; otherwise, we add ρ to the probability that $\Sigma^1 = X$. In the example, the probability that $N_2^1 = \mathsf{Stay}$ would be $p_0(Q^0)$, since Stay appears at level 2 only if the production node $P_1^1 = \langle 0, 1\rangle$, whose probability we have already computed.

The definition of termination determines the values of the termination probabilities, thus completing the specification of the initial belief state B^1. In the example, termination takes place only when $P_1^1 = \langle 4, 1\rangle$; otherwise, additional symbols remain unexpanded on the right-hand side. Therefore, the probability that T_1^1 (and T_1^1 given that $N_1^1 =$Drive) is true is $p_4(Q^0)$. We know that T_2^1 is true with probability zero when $N_2^1 =$Pass, because all of the passing maneuvers involve two steps. However, $N_2^1 =$ nil with probability $1 - p_3(Q^0)$,

INITIALIZE-BELIEF-STATE($grammar, B, d, q$)

$B^1_Q \leftarrow \pi_0(q)$

$B^1_N(1, S) \leftarrow 1.0$

$B^1_N(1, \mathsf{nil}) \leftarrow 0.0$

for $\ell \leftarrow 2$ **to** d

/* Production and symbol beliefs */

for each symbol X such that $B^1_N(\ell - 1, X) > 0.0$

for each production $a = X \rightarrow Y_1 \cdots Y_m$ (p)

$B^1_P(\ell - 1, \langle a, 1\rangle) \leftarrow B^1_N(\ell, X) \Pr(q)$

if $Y_1 \in N$

then $B^1_N(\ell, Y_1)$ += $B^1_P(\ell - 1, \langle a, 1\rangle)$

else $B^1\Sigma(Y_1)$ += $B^1_P(\ell - 1, \langle a, 1\rangle)$

COMPUTE-T($grammar, B, d, 1$)

Figure 5.10: Pseudocode for initializing PSDG belief state under the assumption of completely observable states.

in which case T^1_2 is true with probability one, so the marginal probability that T^1_2 is true is $1-p_3(Q^0)$. Figure 5.10 provides a pseudocode representation of the belief state initialization algorithm.

Computation of Intermediate State Probabilities Given the belief state at time $t-1$, we can compute the desired prediction probabilities over plan events at time t, posterior explanation probabilities over plan events at time $t-1$, and the new belief state for time t. For all three sets of probabilities, we first need to compute a probability distribution over the observed Q^{t-1} given the past evidence. We can easily compute the probability of the observed state conditioned on possible values of Σ^{t-1} using the transition probability function:

$$\Pr(Q^{t-1} = q_{t-1} | \mathcal{E}^{t-3}, Q^{t-2} = q_{t-2}, \Sigma^{t-1} = x) = \pi_1(q_{t-2}, x, q_{t-1}) \tag{5.1}$$

We can then use this result to compute posterior probabilities over Q_t:

$$\Pr(Q^{t-1} = q_{t-1} | \mathcal{E}^{t-3}, Q^{t-2} = q_{t-2}) = \sum_{x \in \Sigma} \pi_1(q_{t-2}, x, q_{t-1}) B^{t-1}_\Sigma(x) \tag{5.2}$$

With the example observations of Figure 5.2, we would thus first compute the probability of the observed state Q^1 given the initial state Q^0 and a possible terminal symbol, either Left, Right, Stay, or Exit. Once we had these four probabilities, we can determine the probability of the observed Q^1 given only Q^0 using Equation 5.2. In general, the time complexity of computing this probability distribution is $O(|\Sigma| \cdot |Q|)$.

Once we observe Q^{t-1} and compute the probability of the evidence Q^{t-1} given each possible terminal symbol (from Equation 5.1), we can then proceed bottom-up through the subplan hierarchy to compute the probability of the evidence conditioned on the possible states of the nonterminal symbol nodes, similar to a generalization of the transition probability function π_1. These probability values are reused many times in subsequent computations, so we define the function π_2^t to simplify notation:

$$\pi_2^t(\ell, X, T) \equiv \Pr(Q^t | \mathcal{E}^{t-1}, N_\ell^t = X, T_\ell^t) \tag{5.3}$$

$$\pi_2^t(\ell, X, \neg T) \equiv \Pr(Q^t | \mathcal{E}^{t-1}, N_\ell^t = X, \neg T_\ell^t) \tag{5.4}$$

In our simple traffic example, $\pi_2^1(2, \mathsf{Pass}, \neg T)$ corresponds to the probability of the observed state Q^1 given that a passing maneuver occurred in state Q^0 without terminating.

We can compute such probabilities recursively by starting with the base definition for all terminal symbols $x \in \Sigma$, for which the function definition reduces to simply the transition probability:

$$\pi_2^t(\ell, x, T) = \pi_1(Q^{t-1}, x, Q^t) \tag{5.5}$$

$$\pi_2^t(\ell, x, \neg T) = 0 \tag{5.6}$$

We can build up the values for nonterminal symbols $X \in N$ by considering all the possible values $\langle a = (X \to Y_1 \cdots Y_m), b \rangle$ at P_ℓ^{t-1}, restricting ourselves to the case where $b = m$ when conditioning on termination:

$$\pi_2^t(\ell, X, T) = \frac{\sum_{\langle a,m \rangle} B_P^t(\ell, \langle a, m \rangle) \pi_2^t(\ell + 1, Y_m, T) B_{T|N}^t(\ell + 1, Y_m)}{B_{T|N}^t(\ell, X) B_N^t(\ell, X)} \tag{5.7}$$

We can compute Equation 5.7 using only values from our belief state B^t and values of π_2 already computed for child symbols. For $Y_m \in \Sigma$, we take $B_{T|N}^t(\ell + 1, Y_m) = 1$.

In our example, $\pi_2^1(2, \mathsf{Pass}, T)$ must consider the two possible terminating production states $\langle 5, 2 \rangle$ and $\langle 6, 2 \rangle$. For each case, Y_m is a terminal symbol, so the numerator of Equation 5.7 is $B_P^1(2, \langle 5, 2 \rangle) \pi_1(Q^0, \mathsf{Left}, Q^1) + B_P^1(2, \langle 6, 2 \rangle) \pi_1(Q^0, \mathsf{Right}, Q^1)$, representing our current beliefs in the two production states, weighted by the likelihood of the observed state

following the execution of the specified action. The denominator simply normalizes the function over the possible nonterminal symbols terminating at that level of the hierarchy.

We also need the probability conditioned on nontermination:

$$\begin{aligned}\pi_2^t(\ell, X, \neg T) &= \left[\sum_{\langle a,b \leq m \rangle} B_P^t(\ell, \langle a,b \rangle) \pi_2^t(\ell+1, Y_b, \neg T) \left(1 - B_{T|N}^t(\ell+1, Y_b)\right) \right. \\ &\quad \left. + \sum_{\langle a,b < m \rangle} B_P^t(\ell, \langle a,b \rangle) \pi_2^t(\ell+1, Y_b, T) B_{T|N}^t(\ell+1, Y_b) \right] \\ &\quad / \left[\left(1 - B_{T|N}^t(\ell, X)\right) B_N^t(\ell, X) \right] \end{aligned} \tag{5.8}$$

For $Y_b \in \Sigma$, we again take $B_{T|N}^t(\ell, Y) = 1$.

In the traffic example, there are two possible productions a involved in $\pi_2^1(2, \mathsf{Pass}, \neg T)$. For production 5, the first summation involves two possible values for b. However, both symbols on the right-hand side are terminal symbols, so $B_{T|N}^1(3, Y_b) = 1$, and the overall summation is zero. In the second summation, we only consider the case when the observed driver is in the first step of the passing maneuver, so $b = 1$. In this case, $Y_b = \mathsf{Left}$ is a terminal symbol, so the summation reduces to $B_P^t(2, \langle 2,1 \rangle) \pi_1(Q^0, \mathsf{Left}, Q^1)$, which is then combined with the normalizing denominator for the final result. At each level of the hierarchy ℓ, computing the expressions of Equations 5.7 and 5.8 requires time $O(|P|m)$, where m is the maximum production length. If d is the depth of the hierarchy, this portion of the dynamic programming phase takes time $O(|P|md)$.

We also need to compute the probability of the evidence conditioned on termination at a particular level of the hierarchy, regardless of the production or symbol:

$$\begin{aligned}\Pr(Q^{t-1} | \mathcal{E}^{t-2}, T_\ell^{t-1}) &= \left[\sum_{X \in N} B_N^{t-1}(\ell, X) \pi_2^{t-1}(\ell, X, T) B_{T|N}^{t-1}(\ell, X) + \right. \\ &\quad \left. \sum_{\langle a,b \rangle | Y_b \in \Sigma} \sum_{k=0}^{\ell-1} B_P^{t-1}(k, \langle a,b \rangle) \pi_1(Q^{t-2}, Y_b, Q^{t-1}) \right] / B_T^{t-1}(\ell) \end{aligned} \tag{5.9}$$

Computing the first summation of Equation 5.9 requires time $O(|N|)$, while the second, which handles the case where $N_\ell^{t-1} = \mathsf{nil}$, requires time $O(|P|m\ell)$, for a given level of the hierarchy ℓ. At level 2 of our example hierarchy, the first summation has only Pass to consider. The second summation must consider production states $\langle 0,1 \rangle$, $\langle 1,1 \rangle$, $\langle 2,1 \rangle$, and $\langle 4,1 \rangle$ at level 1 of the hierarchy, because all of these production states produce terminal symbols, making $N_2^1 = \mathsf{nil}$. To compute the values over the entire hierarchy requires time $O(|P|md^2)$, which dwarfs the time complexity of the first portion of the dynamic programming phase

(Equations 5.7 and 5.8). Figure 5.11 provides a pseudocode description of the dynamic programming algorithms for computing π_2 and the other intermediate probabilities, as well as the explanation probabilities described in the next subsection.

Computation of Explanation Probabilities We can use these dynamic programming results to obtain the posterior probability distribution over symbols and productions at time $t-1$ conditioned on evidence up to and including time t. This probability distribution is useful for answering explanation queries, where we want to interpret the agent's past behavior in light of the new evidence. For a given level of the hierarchy, there are two possibilities for termination of the current child. In the following expression, we can ignore the nonterminating case if a terminal symbol is involved:

$$\begin{aligned}\Pr(N_\ell^{t-1} = X|\mathcal{E}^{t-1}) &= B_N^{t-1}(\ell, X)\left[\pi_2^{t-1}(\ell, X, T)B_{T|N}^{t-1}(\ell, X)\right. \\ &\quad \left.+\pi_2^{t-1}(\ell, X, \neg T)\left(1 - B_{T|N}^{t-1}(\ell, X)\right)\right]/B_Q^t \end{aligned} \tag{5.10}$$

Each such posterior probability is computed in constant time since all the quantities involved are stored in our belief states, including B_Q^t which we compute using Equation 5.2.

In our example, we can use Equation 5.2 to compute the probability of a passing maneuver given our observations of Q^0 and Q^1. The first term within the brackets is zero, because it is impossible that a Pass took place at time 1 and terminated, so $B_{T|N}^1(2, \mathsf{Pass}) = 0$. With only the second term remaining, the entire expression reduces to $B_N^1(2, \mathsf{Pass})\pi_2^1(2, \mathsf{Pass}, \neg T)/B_Q^2$, which we can compute immediately given the belief states for times 1 and 2, as well as our dynamic programming results. Such probabilities can form the basis for a recognizing agent's interpretation of the observed agent's past behavior.

It will turn out to be useful to have the probabilities $\Pr(T_\ell^{t-1}|\mathcal{E}^{t-1}, T_{\ell+1}^{t-1})$, the posterior probability of termination at level ℓ given termination at the lower level $\ell+1$:

$$\begin{aligned}\Pr(T_\ell^{t-1}|\mathcal{E}^{t-1}, T_{\ell+1}^{t-1}) &= \frac{\sum_{\langle a,m\rangle} B_P^{t-1}(\ell, \langle a, m\rangle)\pi_2^{t-1}(\ell+1, Y_m, T)B_{T|N}^{t-1}(\ell+1, Y_m)}{\Pr(Q^{t-1}|\mathcal{E}^{t-2}, T_{\ell+1}^{t-1})B_T^{t-1}(\ell+1)} \\ &\quad + \frac{B_N^{t-1}(\ell, \mathsf{nil})}{B_T^{t-1}(\ell+1)} \end{aligned} \tag{5.11}$$

In the traffic example, we would compute $\Pr(T_1^1|Q^0, Q^1, T_2^1)$ by considering all of the productions expanding Drive. Productions 0, 1, and 2 are recursive, so we can never be in the production state indicating the last symbol (where the recursion occurs). The form of recursion discussed in Section 5.2.2 requires that, once we finish expanding the symbol before the recursion, we would instead assign the production state to be a new expansion

EXPLANATION-PHASE($grammar, B, d, q_0, q_1, t$)
  /* Compute probability of evidence */
  for each $x \in \Sigma$
    B_Q^t += $\pi_1(q_0, x, q_1) B_\Sigma^{t-1}(x)$
  /* Compute values of π_2 */
  for $\ell \leftarrow d$ **down-to** 1
    for each symbol X such that $B_N^t(\ell, X) > 0.0$
      for each production $a = X \rightarrow Y_1 \cdots Y_m$ (p)
        if $Y_m \neq X$
          if $Y_m \in N$
            then $\pi_2^{t-1}(\ell, X, T)$ += $B_P^{t-1}(\ell, \langle a, m \rangle) \pi_2^{t-1}(\ell + 1, Y_m, T) B_{T|N}^{t-1}(\ell + 1, Y_m)$
              $\pi_2^{t-1}(\ell, X, \neg T)$ += $B_P^{t-1}(\ell, \langle a, m \rangle) \pi_2^{t-1}(\ell + 1, Y_m, \neg T)$
                $(1 - B_{T|N}^{t-1}(\ell + 1, Y_m))$
            else $\pi_2^{t-1}(\ell, X, T)$ += $B_P^{t-1}(\ell, \langle a, m \rangle) \pi_1(q_0, Y_m, q_1)$
          for $b \leftarrow 1$ **to** $m - 1$
            if $Y_b \in N$
              then $\pi_2^{t-1}(\ell, X, \neg T)$ += $B_P^{t-1}(\ell, \langle a, b \rangle) \pi_2^{t-1}(\ell + 1, Y_b, \neg T)$
                $(1 - B_{T|N}^{t-1}(\ell + 1, Y_b))$
                $\pi_2^{t-1}(\ell, X, \neg T)$ += $B_P^{t-1}(\ell, \langle a, b \rangle) \pi_2^{t-1}(\ell + 1, Y_b, T) (B_{T|N}^{t-1}(\ell + 1, Y_b))$
              else $\pi_2^{t-1}(\ell, X, \neg T)$ += $B_P^{t-1}(\ell, \langle a, b \rangle) \pi_1^{t-1}(q_0, Y_b, q_1)$
        $\pi_2(\ell, X, T)$ /= $B_{T|N}^{t-1}(\ell, X) B_N^{t-1}(\ell, X)$
        $\pi_2^{t-1}(\ell, \neg T)$ /= $(1 - B_{T|N}^{t-1}(\ell, X)) B_N^{t-1}(\ell, X)$
        /* Compute values of $\Pr(Q^{t-1} | \mathcal{E}^{t-2}, T_\ell^{t-1})$ */
        $\Pr(Q^{t-1} = q_1 | \mathcal{E}^{t-2}, T_\ell^{t-1})$ += $B_N^{t-1}(\ell, X) \pi_2^{t-1}(\ell, X, T) B_{T|N}^{t-1}(\ell, X)$
      for each $k \leftarrow 1$ **to** $\ell - 1$
        for each $\langle a, b \rangle$ such that $Y_b \in \Sigma$
          $\Pr(Q^{t-1} = q_1 | \mathcal{E}^{t-2}, T_\ell^{t-1})$ += $B_P^{t-1}(k, \langle a, b \rangle) \pi_q(q_0, Y_b, q_1)$
    $\Pr(Q^{t-1} = q_1 | \mathcal{E}^{t-2}, T_\ell^{t-1})$ /= $B_T^{t-1}(\ell)$
    /* Compute values of $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1})$ */
    for each production a with m symbols on the right-hand side
      if $Y_m \in \Sigma$
        then $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1})$ += $B_P^{t-1}(\ell, \langle a, m \rangle) \pi_1(q_0, Y_m, q_1)$
        else $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1})$ += $B_P^{t-1}(\ell, \langle a, m \rangle) \pi_2^{t-1}(\ell + 1, Y_m, T) B_{T|N}^{t-1}(\ell + 1, Y_m)$
    if $\ell <$ **then** $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1})$ /= $\Pr(Q^{t-1} | \mathcal{E}^{t-2}, T_{\ell+1}^{t-1}) B_T^{t-1}(\ell + 1)$
      $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1})$ += $B_N^{t-1}(\ell, \mathsf{nil}) / B_T^{t-1}(\ell + 1)$
      $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1})$ $\leftarrow$ $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1}) B_T^{t-1}(\ell + 1)$
    else $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1})$ /= B_Q^t
    $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1})$ += $B_N^{t-1}(\ell, \mathsf{nil})$

Figure 5.11: Pseudocode for computing explanation probabilities under the assumption of completely observable states.

of Drive. Thus, $B_P^1(1, \langle a, 2\rangle) = 0$ when $a = 0$, 1, or 2. This belief is zero for $a = 3$ as well, since we cannot be in the second stage of a passing maneuver at time 1. Therefore, the only nonzero term corresponds to the exiting maneuver, where the numerator of the first term is $B_P^1(1, \langle 4, 1\rangle)\, \pi_1(Q^0, \mathsf{Exit}, Q^1)$. The second term is zero, because $N_1^1 = \mathsf{Drive}$ with probability one.

Notice that from these probabilities, we can compute the necessary belief state probabilities $\Pr(T_\ell^{t-1}|\mathcal{E}^{t-1})$ in the same bottom-up iteration. For $\ell = d$, this probability is identical to the probability obtained from Equation 5.11, and for all other ℓ:

$$\Pr(T_\ell^{t-1}|\mathcal{E}^{t-1}) = \Pr(T_\ell^{t-1}|\mathcal{E}^{t-1}, T_{\ell+1}^{t-1}) B_T^{t-1}(\ell + 1) \tag{5.12}$$

In the traffic example, the production at level 2 of the hierarchy only terminates if $N_2^1 = \mathsf{nil}$; otherwise, we must be in the first stage of a passing maneuver. To compute $\Pr(T_1^1|Q^0)$, we multiply our result from Equation 5.11 with our belief in termination at level 2, $B_T^1(2)$.

We can compute all of these posterior termination probabilities in one bottom-up pass through the hierarchy requiring time $O(d|P|)$. Figure 5.11 includes a pseudocode description for this algorithm for the computation of the posterior termination probabilities. The pseudocode omits the computation of the posterior symbol probabilities, which a recognizing agent computes only as needed, and which it can directly compute using Equation 5.10.

Computation of Prediction Probabilities For plan prediction at time $t > 0$, assume that we have computed all of the explanation probabilities over the symbols and productions at time $t - 1$. For a production node value of $\langle a, b\rangle$, we denote the production probability function for a as p_a. We first consider the case where $b = 1$, i.e. we are in the first stage of the expansion. This is the only case where the value of the production node is not completely determined by the belief state of the previous time slice. The first term covers the case where we have chosen the production in the current time slice, the second the case where we are simply carrying over the value from the previous time slice because the child had not terminated, and the third the case where we have chosen the production in the current time slice through recursion:

$$\begin{aligned} B_P^t(\ell, \langle a, 1\rangle) \;=\; & p_a(Q^{t-1}) \Pr(N_\ell^t = X|\mathcal{E}^{t-1}, T_\ell^{t-1}) B_T^{t-1}(\ell) + \\ & B_P^{t-1}(\ell, \langle a, 1\rangle) \pi_2^{t-1}(\ell + 1, Y_1, \neg T) \left(1 - B_{T|N}^{t-1}(\ell + 1, Y_1)\right) / B_Q^t + \\ & p_a(Q^{t-1}) \sum_{\langle a,m\rangle | Y_m = X} B_P^{t-1}(\ell, \langle a, m-1\rangle) \pi_2^{t-1}(\ell + 1, Y_{m-1}, T) \cdot \\ & B_{T|N}^{t-1}(\ell + 1, Y_{m-1}) / B_Q^t \end{aligned} \tag{5.13}$$

For instance, upon observing the car in Figure 5.2 move into the leftmost lane, the recognizing agent can determine the likelihood that the driver will now choose to stay in the left lane. Only the third term of Equation 5.13 is nonzero for the belief of interest, $B_P^2(1, \langle 0, 1\rangle)$, since production 0 can fire at time 2 only if recursion takes place. The third term involves the production probability function p_0 and a summation over productions 0, 1, and 2, although our belief in productions 0 and 2 should be zero upon observing the left lane change. Production 3 also involves recursion, but $B_{T|N}^1(2, \mathsf{Pass}) = 0.0$ since the passing maneuver cannot have terminated at time 1. Production 4 does not involve recursion, so the overall production probability reduces to $p_0(Q^1)B_P^1(1, \langle 1, 1\rangle)\pi_1(Q^0, \mathsf{Left}, Q^1)/B_Q^2$.

If we are in the middle of an expansion, we no longer have to worry about matching the current state against the conditions of the production, because the current value is completely determined. There are two cases, depending on whether the child symbol's expansion had terminated or not:

$$\begin{aligned} B_P^t(\ell, \langle a, b > 1\rangle) \;=\; & \Big[B_P^{t-1}(\ell, \langle a, b\rangle)\pi_2^{t-1}(\ell+1, Y_b, \neg T)\left(1 - B_{T|N}^{t-1}(\ell+1, Y_b)\right) + \\ & B_P^{t-1}(\ell, \langle a, b-1\rangle)\pi_2^{t-1}(\ell+1, Y_{b-1}, T)B_{T|N}^{t-1}(\ell+1, Y_{b-1})\Big] \,/ \\ & B_Q^t \end{aligned} \tag{5.14}$$

The recognizing agent in our traffic example can use Equation 5.14 to determine whether the observed driver will perform the second stage of a passing maneuver at time 2. The first term is zero for the desired $B_P^2(1, \langle 3, 2\rangle)$, because it is impossible that the driver had begun performing the second stage at time 1, so $B_P^1(1, \langle 3, 2\rangle) = 0.0$. The overall belief thus reduces to $B_P^1(1, \langle 3, 1\rangle)\pi_1(Q^0, \mathsf{Left}, Q^1)/B_Q^2$.

All of the symbol variables have deterministic relationships on other random variables. The top-level symbol variable N_1^t is nil only if the top-level expansion of the start symbol has terminated at the previous time slice:

$$B_N^t(1, S) = 1 - \Pr(T_1^{t-1} | \mathcal{E}^{t-1}) \tag{5.15}$$

The probability on the right-hand side is the posterior probability of termination at the top level, and we have already computed this quantity during the explanation phase. All other symbol variables are determined by the production variables at the level above, since if the parent production has a value of $\langle a, b\rangle$, where production a has a right-hand side $Y_1 \cdots Y_m$, then the child symbol must be Y_b:

$$B_N^t(\ell > 1, X) \;=\; \sum_{\langle a,b\rangle | Y_b = X} B_P^t(\ell - 1, \langle a, b\rangle) \tag{5.16}$$

$$B^t_\Sigma(x) \quad = \quad \sum_\ell \sum_{\langle a,b\rangle | Y_b = x} B^t_P(\ell, \langle a, b\rangle) \tag{5.17}$$

For instance, the recognizing driver could compute the probability that the observed driver stays in the leftmost lane in time 2 by querying the symbol probability, $B^2_\Sigma(\mathsf{Stay})$. In this case, there is only one possible production state that produces a Stay action, $B^2_P(1, \langle 0, 1\rangle)$.

Equation 5.13 uses $\Pr(N^t_\ell = X | \mathcal{E}^{t-1}, T^{t-1}_\ell)$, the probability of symbol X in the current time slice, given termination, in the previous time slice, of the production at the same level. We compute such probabilities top-down, beginning with the start symbol, which has probability zero if its production terminated in the previous time slice:

$$\Pr(N^t_1 = S | \mathcal{E}^{t-1}, T^{t-1}_1) = 0 \tag{5.18}$$

We then proceed through the hierarchy finding the probability over production states given termination of its child in the previous time slice. The formula is similar to that for the unconditional production probability (Equations 5.13 and 5.14), except that we no longer include the nonterminating cases:

$$\begin{aligned} & \Pr(P^t_\ell = \langle a = X \to \xi, 1\rangle \, | \mathcal{E}^{t-1}, T^{t-1}_{\ell+1}) \\ = \quad & p_a(Q^{t-1}) \Pr(N^t_\ell = X | \mathcal{E}^{t-1}, T^{t-1}_\ell) \Pr(T^{t-1}_\ell | \mathcal{E}^{t-1}, T^{t-1}_{\ell+1}) + \\ & \sum_{\langle a,m\rangle | Y_m = X} p_a(Q^{t-1}) \pi^{t-1}_2(\ell + 1, Y_{m-1}, T) B^{t-1}_P(\ell, \langle a, m-1\rangle) / B^t_Q \end{aligned} \tag{5.19}$$

We have already computed the required probability $\Pr(T^{t-1}_\ell | \mathcal{E}^{t-1}, T^{t-1}_{\ell+1})$ during the explanation phase. For intermediate production states:

$$\begin{aligned} \Pr(P^t_\ell = \langle a, b > 1\rangle \, | \mathcal{E}^{t-1}, T^{t-1}_{\ell+1}) \quad = \quad & \pi^{t-1}_2(\ell + 1, Y_{b-1}, T) B^{t-1}_P(\ell, \langle a, b-1\rangle) B^{t-1}_{T|N}(\ell + 1, Y_{b-1}) \\ & / \Pr(Q^{t-1} | \mathcal{E}^{t-2}, T^{t-1}_{\ell+1}) \end{aligned} \tag{5.20}$$

Again, we have already computed the probability $\Pr(Q^{t-1} | \mathcal{E}^{t-2}, T^{t-1}_{\ell+1})$ during the explanation phase. We can use these production probabilities to compute the desired distribution over symbols, conditioned on production termination during previous time slice:

$$\Pr(N^t_\ell = X, \ell > 1 | \mathcal{E}^{t-1}, T^{t-1}_\ell) = \sum_{\langle a,b\rangle | Y_b = X} \Pr(P^t_{\ell-1} = \langle a, b\rangle \, | \mathcal{E}^{t-1}, T^{t-1}_\ell) \tag{5.21}$$

The computation of symbol probabilities conditioned on termination occurs coincidentally with the computation of the unconditional prediction probabilities and incurs no additional time cost, since it performs no new computations. In computing the unconditional

probabilities, we have to compute Equation 5.13 once for each production at each level of the hierarchy, and each such computation takes constant. In addition, we have to compute Equation 5.14 $O(m)$ times for each production at each level of the hierarchy, but each computation takes only constant time, using only probabilities already stored in the grammar, belief state, or the π_2 function. Therefore, the time complexity of the computing the probabilities of all production states is $O(d|P|m)$. Since each production state contributes to the summation for exactly one symbol $X \in N$, we can compute these symbol probabilities within the same algorithm for computing the production probabilities. Thus, we incur no additional time cost in computing symbol probabilities beyond that incurred for the production probabilities. Figure 5.12 presents a pseudocode description of these algorithms for the computation of prediction probabilities.

Computation of New Belief State The prediction phase specifies most of the components of the belief state B^t. It remains only to specify the termination components. It is straightforward, from the definition of termination, to compute the required probability of termination given the symbol in a single bottom-up pass through the hierarchy:

$$B^t_{T|N}(\ell, X) = \sum_{\langle a=X\rightarrow\cdots,m\rangle} B^t_P(\ell, \langle a, m\rangle) B^t_{T|N}(\ell+1, Y_m) / B^t_N(\ell, X) \tag{5.22}$$

In the traffic example, we can compute the probability that the production at level 2 of the hierarchy terminates at time 2, given that the observed driver is performing a passing maneuver. Having already eliminated production 6 from possibility, we reduce belief $B^2_{T|N}(2, \mathsf{Pass})$ to $B^2_P(2, \langle 5, 2\rangle)/B^2_N(2, \mathsf{Pass})$.

We can use this conditional result to compute the termination probability independent of symbol:

$$B^t_T(\ell) = \left(\sum_{X\in N} B^t_{T|N}(\ell, X) B^t_N(\ell, X)\right) + B^t_N(\ell, \mathsf{nil}) \tag{5.23}$$

We can compute these probabilities in a single bottom-up pass through the hierarchy requiring time $O(d|P|)$. Figure 5.13 provides a pseudocode description of this termination probability algorithm.

Overall, the explanation, prediction, and belief revision algorithms for a single time step have time complexity $O(|\Sigma| \cdot |Q| + d|P|m)$. If we do not compute a probability distribution over the future state, we ignore the first term and the time complexity is simply $O(d|P|m)$. The revised belief state B^t is sufficient for answering queries about the symbols at time

PREDICTION-PHASE($grammar, B, d, q, t$)

$B_N^t(1, S) \leftarrow 1.0 - \Pr(T_1^{t-1} | \mathcal{E}^{t-1})$

$B_N^t(1, \mathsf{nil}) \leftarrow 1.0 - B_N^t(1, S)$

for $\ell \leftarrow 2$ **to** d

/* Production and symbol beliefs, including beliefs conditioned on termination */

for each symbol X such that $B_N^t(\ell - 1, X) > 0.0$

for each production $a = X \rightarrow Y_1 \cdots Y_m$ (p)

$B_P^t(\ell - 1, \langle a, 1 \rangle) \leftarrow p_a(Q^{t-1}) \Pr(N_{\ell-1}^t = X | \mathcal{E}^{t-1}, T_{\ell-1}^{t-1}) B_T^{t-1}(\ell - 1) +$

$\pi_2^{t-1}(\ell, Y_1, \neg T) \left(1 - B_{T|N}^{t-1}(\ell, Y_1)\right) B_P^{t-1}(\ell - 1, \langle a, 1 \rangle) / B_Q^t +$

$p_a(Q^{t-1}) \sum_{\langle a,m \rangle | Y_m = X} \pi_2^{t-1}(\ell, Y_{m-1}, T) B_{T|N}^{t-1}(\ell, Y_{m-1}) \cdot$

$B_P^{t-1}(\ell - 1, \langle a, m - 1 \rangle) / B_Q^t$

$\Pr(P_{\ell-1}^t = \langle a, 1 \rangle | \mathcal{E}^{t-1}, T_\ell^{t-1}) \leftarrow p_a(Q^{t-1}) \Pr(N_{\ell-1}^t = X | \mathcal{E}^{t-1}, T_{\ell-1}^{t-1})$

$\Pr(T_{\ell-1}^{t-1} | \mathcal{E}^{t-1}, T_\ell^{t-1}) + p_a(Q^{t-1}) \sum_{\langle a,m \rangle | Y_m = X} \cdot$

$\pi_2^{t-1}(\ell, Y_{m-1}, T) B_{T|N}^{t-1}(\ell, Y_{m-1}) B_P^{t-1}(\ell - 1, \langle a, m - 1 \rangle) / B_Q^t$

for $b \leftarrow 2$ **to** m

$B_P^t(\ell - 1, \langle a, b \rangle) \leftarrow \Big[\pi_2^{t-1}(\ell, Y_b, \neg T) \left(1 - B_{T|N}^{t-1}(\ell, Y_b)\right) \cdot$

$B_P^{t-1}(\ell - 1, \langle a, b \rangle) +$

$\pi_2^{t-1}(\ell, Y_b, T) B_{T|N}^{t-1}(\ell, Y_b) B_P^{t-1}(\ell - 1, \langle a, b - 1 \rangle)\Big] / B_Q^t$

$\Pr(P_{\ell-1}, \langle a, b \rangle | \mathcal{E}^{t-1}, T_\ell^{t-1}) \leftarrow$

$\Big[\pi_2^{t-1}(\ell, Y_b, \neg T) \left(1 - B_{T|N}^{t-1}(\ell, Y_b)\right) B_P^{t-1}(\ell - 1, \langle a, b \rangle) +$

$\pi_2^{t-1}(\ell, Y_b, T) B_{T|N}^{t-1}(\ell, Y_b) B_P^{t-1}(\ell - 1, \langle a, b - 1 \rangle)\Big] / B_Q^t$

for $b \leftarrow 1$ **to** m

if $Y_b \in N$

then $B_N^t(\ell, Y_b)$ += $B_P^t(\ell - 1, \langle a, b \rangle)$

$\Pr(N_\ell = Y_b | \mathcal{E}^{t-1}, T_\ell^{t-1})$ += $\Pr(P_{\ell-1} = \langle a, b \rangle | \mathcal{E}^{t-1}, T_\ell^{t-1})$

else $B_\Sigma^t(Y_b)$ += $B_P^t(\ell - 1, \langle a, b \rangle)$

COMPUTE-T($grammar, B, d, t$)

Figure 5.12: Pseudocode for computing prediction probabilities under the assumption of completely observable states.

COMPUTE-T($grammar, B, d, t$)

for $\ell \leftarrow d-1$ **down-to** 1

for each symbol X such that $B_N^t(\ell, X) > 0.0$

for each production $a = X \rightarrow Y_1 \cdots Y_m$ (p)

if $Y_m \neq X$

if $Y_m \in N$

then $B_{T|N}^t(\ell, X)$ += $B_P^t(\ell, \langle a, m \rangle) B_{T|N}^t(\ell+1, Y_m) / B_N^t(\ell, X)$

else $B_{T|N}^t(\ell, X)$ += $B_P^t(\ell, \langle a, m \rangle) / B_N^t(\ell, X)$

$B_T^t(\ell)$ += $B_{T|N}^t(\ell, X) B_N^t(\ell, X)$

Figure 5.13: Pseudocode for computing termination probabilities under the assumption of completely observable states.

t, as well as supporting the computation of future belief states, so we no longer need the probability values stored in belief state B^{t-1}.

Partially Observable States

We now relax the assumption that we observe the value of the state variable Q^t. Instead, we observe that $Q^t \in R^t$, for some subset $R_t \subseteq Q$. In most cases, this subset is the set of state elements that are consistent with the observed features. For instance, in our traffic example, we can observe the position and speed of the observed car, as well as the position and speeds of the other cars on the highway. However, we cannot observe aspects of the observed driver's mental state. In this case, R_t is the set of state values ranging over the possible unobserved state features and consistent with the observed features.

Our evidence variable $\mathcal{E}^t$ now becomes the sequence of responses $Q^0 \in R^0, \ldots, Q^t \in R^t$. When we knew the exact value of Q^t, production choices at time $t+1$ were either chosen based on Q^t (if termination occurred at time t), or determined by the value of the production node at time t (if termination did not occur). We no longer know the exact value of Q^t, so in the terminating case, the production choice for time $t+1$ is dependent on the parents of Q^t, which are the terminal symbol variable Σ^t and the previous state variable Q^{t-1}. However, because we no longer now the exact value of Q^{t-1} either, we must also consider its dependency in turn on Σ^{t-1} and Q^{t-2}, which is also unknown. In the end, we must

record the entire sequence of observations to have sufficient information to compute any exact symbol probabilities.

We clearly need a different belief structure to avoid the unreasonable space requirements of such an approach. We can regain the independence that simplified the algorithms for completely observable states by focusing on each point in R^t separately. In other words, instead of a single belief state conditioned on the evidence, we maintain separate belief states for each possible value for $Q^t \in R^t$. In the traffic example, there would be a unique belief state corresponding to the distribution over events given a particular configuration of the unobserved mental state. We then add a corresponding argument to all of the functions of Figure 5.9. For instance, $B_N^t(\ell, X, q)$ would return the probability $\Pr(N_\ell^t = X|\mathcal{E}^{t-2}, Q^{t-1} = q)$. This new belief state has a space complexity of $O(|R^t|d|P|m)$.

We can modify the inference algorithms for completely observable state to consider all of the possible state values within R^{t-1} (and R^{t-2} for the explanation phase). For instance, we can initialize the set of belief states by calling INITIALIZE-BELIEF-STATE($grammar, B(q), d, q$), for all $q \in R^0$. Figures 5.14 and 5.15 present pseudocode descriptions of similarly modified algorithms for performing the same explanation and prediction tasks as the algorithms for completely observable states. We call both for all state values R^{t-1}. The algorithm for COMPUTE-T is identical to that from Figure 5.13, except for the additional q argument (again ranging through all values in R^{t-1}) introduced into the belief state, as well as into the procedure itself.

Because the belief states are conditioned on individual points of the state space, they no longer have the prediction probabilities stored explicitly. However, we can easily obtain the marginal probabilities wanted for plan recognition queries. For instance:

$$\Pr(N_\ell^t = X|\mathcal{E}^{t-1}) = \sum_{q \in R^{t-1}} B_N^t(\ell, X, q) B_Q^t(q) \tag{5.24}$$

The time complexity of the inference algorithms for time t with partial observations is $O(|R^{t-1}|^2 d|P|m)$.

5.2.4 Implementation of PSDG Algorithms

The implemented algorithms of Section 5.2.3 supported practical inference in two domains, described in Section 6.2. With a PSDG representation of a given problem domain, a recognizing agent can perform its own decision process based on a probability distribution over the possible subplans and actions of the observed agent. The compact belief state

EXPLANATION-PHASE($grammar, B, d, q_1, t$)
/* Compute probability of evidence */
for each $q_0 \in R^{t-2}$
for each $x \in \Sigma$
$\Pr(Q^{t-1} = q_1 | \mathcal{E}^{t-3}, Q^{t-2} = q_0)$ += $\pi_1(q_0, x, q_1) B_\Sigma^{t-1}(x, q_0)$
$B_Q^t(q_0)$ += $\Pr(Q^{t-1} = q_1 | \mathcal{E}^{t-3}, Q^{t-2} = q_0) B_Q^{t-2}(q_0)$
/* Compute values of π_2 */
for $\ell \leftarrow d$ **down-to** 1
for each symbol X such that $B_N^t(\ell, X) > 0.0$
for each production $a = X \rightarrow Y_1 \cdots Y_m$ (p)
if $Y_m \neq X$
if $Y_m \in N$
then $\pi_2^{t-1}(q_0, \ell, X, T, q_1)$ +=
$B_P^{t-1}(\ell, \langle a, m \rangle, q_0) \pi_2^{t-1}(q_0, \ell + 1, Y_m, T, q_1) B_{T|N}^{t-1}(\ell + 1, Y_m, q_0)$
$\pi_2^{t-1}(q_0, \ell, X, \neg T, q_1)$ +=
$B_P^{t-1}(\ell, \langle a, m \rangle, q_0) \pi_2^{t-1}(q_0, \ell + 1, Y_m, \neg T, q_1) \cdot$
$(1 - B_{T|N}^{t-1}(\ell + 1, Y_m), q_0)$
else $\pi_2^{t-1}(q_0, \ell, X, T, q_1)$ += $B_P^{t-1}(\ell, \langle a, m \rangle, q_0) \pi_1(q_0, Y_m, q_1)$
for $b \leftarrow 1$ **to** $m - 1$
if $Y_b \in N$
then $\pi_2^{t-1}(q_0, \ell, X, \neg T, q_1)$ +=
$B_P^{t-1}(\ell, \langle a, b \rangle, q_0) \pi_2^{t-1}(q_0, \ell + 1, Y_b, \neg T, q_1)(1 - B_{T|N}^{t-1}(\ell + 1, Y_b), q_0)$
$\pi_2^{t-1}(q_0, \ell, X, \neg T, q_1)$ +=
$B_P^{t-1}(\ell, \langle a, b \rangle, q_0) \pi_2^{t-1}(q_0, \ell + 1, Y_b, T, q_1)(B_{T|N}^{t-1}(\ell + 1, Y_b), q_0)$
else $\pi_2^{t-1}(q_0, \ell, X, \neg T, q_1)$ += $B_P^{t-1}(\ell, \langle a, b \rangle) \pi_1^{t-1}(q_0, Y_b, q_1)$
$\pi_2(q_0, \ell, X, T, q_1)$ /= $B_{T|N}^{t-1}(\ell, X, q_0) B_N^{t-1}(\ell, X, q_0)$
$\pi_2^{t-1}(q_0, \ell, X, \neg T, q_1)$ /= $(1 - B_{T|N}^{t-1}(\ell, X), q_0) B_N^{t-1}(\ell, X, q_0)$
$\Pr(Q^{t-1} = q_1 | \mathcal{E}^{t-3}, Q^{t-2} = q_0, T_\ell^{t-1})$ += $B_N^{t-1}(\ell, X, q_0) \pi_2^{t-1}(q_0, \ell, X, T, q_1) \cdot$
$B_{T|N}^{t-1}(\ell, X, q_0)$
for each $k \leftarrow 1$ **to** $\ell - 1$
for each $\langle a, b \rangle$ such that $Y_b \in \Sigma$
$\Pr(Q^{t-1} = q_1 | \mathcal{E}^{t-3}, Q^{t-2} = q_0, T_\ell^{t-1})$ += $B_P^{t-1}(k, \langle a, b \rangle, q_0) \pi_q(q_0, Y_b, q_1)$
$\Pr(Q^{t-1} = q_1 | \mathcal{E}^{t-3}, Q^{t-2} = q_0, T_\ell^{t-1})$ /= $B_T^{t-1}(\ell, q_0)$
if $Y_m \in \Sigma$
then $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1}, Q^{t-1} = q_1)$ +=
$B_P^{t-1}(\ell, \langle a, m \rangle, q_0) \pi_1(q_0, Y_m, q_1) B_Q^{t-1}(q_0)$
else $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1}, Q^{t-1} = q_1)$ +=
$B_P^{t-1}(\ell, \langle a, m \rangle, q_0) \pi_2^{t-1}(q_0, \ell + 1, Y_m, T, q_1)$
$B_{T|N}^{t-1}(\ell + 1, Y_m, q_0) B_Q^{t-1}(q_0)$
if $\ell < d$ **then** $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-3} Q^{t-1} = q_1, T_{\ell+1}^{t-1})$ /=
$\Pr(Q^{t-1} = q_1 | \mathcal{E}^{t-3}, T_{\ell+1}^{t-1}) B_T^{t-1}(\ell + 1, q_0)$
$\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1}, Q^{t-1} = q_1)$ += $B_N^{t-1}(\ell, \mathsf{nil}, q_0) / B_T^{t-1}(\ell + 1, q_0)$
$\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, Q^{t-1} = q_1)$ += $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-3}, Q^{t-1} = q_1, T_{\ell+1}^{t-1}) \cdot$
$\Pr(T_{\ell+1}^{t-1} | \mathcal{E}^{t-1}, Q^{t-1} = q_1)$
else $\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1}, Q^{t-1} = q_1)$ /= $B_Q^t(q_1)$
$\Pr(T_\ell^{t-1} | \mathcal{E}^{t-1}, T_{\ell+1}^{t-1}, Q^{t-1} = q_1)$ += $B_N^{t-1}(\ell, \mathsf{nil}, q_0)$

Figure 5.14: Pseudocode for explanation phase of PSDG inference algorithm without completely observable states.

PREDICTION-PHASE($grammar, B, d, q_1, t$)
  $B_N^t(1, S, q_1) \leftarrow 1.0 - \Pr(T_1^{t-1} | \mathcal{E}^{t-1}, Q^{t-1} = q_1)$
  $B_N^t(1, \mathsf{nil}, q_1) \leftarrow 1.0 - B_N^t(1, S, q_1)$
  for $\ell \leftarrow 2$ **to** d
    /* Production and symbol beliefs, including beliefs conditioned on termination */
    for each symbol X such that $B_N^t(\ell - 1, X, q_1) > 0.0$
      for each production $a = X \rightarrow Y_1 \cdots Y_m$ (p)
        $B_P^t(\ell - 1, \langle a, 1 \rangle, q_1)$ +=
          $p_a(q_1) \Pr(N_{\ell-1}^t = X | \mathcal{E}^{t-1}, T_{\ell-1}^{t-1}, Q^{t-1} = q_1) B_T^{t-1}(\ell - 1, q_1)$
        for each $q_0 \in R^{t-2}$
          $B_P^t(\ell - 1, \langle a, 1 \rangle, q_1)$ +=
            $\pi_2^{t-1}(q_0, \ell, Y_1, \neg T, q_1) \left(1 - B_{T|N}^{t-1}(\ell, Y_1, q_0)\right) B_P^{t-1}(\ell - 1, \langle a, 1 \rangle, q_0) \cdot$
              $B_Q^{t-1}(q_0) / B_Q^t(q_1)$
          $B_P^t(\ell - 1, \langle a, 1 \rangle, q_1)$ +=
            $p_a(q_1) \sum_{\langle a,m \rangle | Y_m = X} B_P^{t-1}(\ell - 1, \langle a, m - 1 \rangle, q_0) \pi_2^{t-1}(q_0, \ell, Y_{m-1}, T, q_1)$
              $B_{T|N}^{t-1}(\ell, Y_{m-1}, q_0) \Pr(Q^{t-1} = q_1 | \mathcal{E}^{t-3}, Q^{t-2} = q_0) B_Q^{t-1}(q_0) / B_Q^t(q_1)$
          $\Pr(P_{\ell-1}^t = \langle a, 1 \rangle | \mathcal{E}^{t-1}, T_\ell^{t-1}, Q^{t-1} = q_1)$ +=
            $p_a(q_1) \Pr(N_{\ell-1}^t = X | \mathcal{E}^{t-2}, Q^{t-1} = q_1, T_{\ell-1}^{t-1}) \Pr(T_{\ell-1}^{t-1} | \mathcal{E}^{t-1}, T_\ell^{t-1}, Q^{t-1} = q_1)$
            $+ p_a(Q^{t-1}) \sum_{\langle a,m \rangle | Y_m = X} B_P^{t-1}(\ell - 1, \langle a, m - 1 \rangle, q_0) \pi_2^{t-1}(q_0, \ell, Y_{m-1}, T, q_1)$
              $B_{T|N}^{t-1}(\ell, Y_{m-1}, q_0) \Pr(Q^{t-1} = q | \mathcal{E}^{t-3}, Q^{t-2} = q_0) B_Q^{t-1}(q_0) / B_Q^t(q_1)$
          for $b \leftarrow 2$ **to** m
            $B_P^t(\ell - 1, \langle a, b \rangle)$ +=
              $\Big[\pi_2^{t-1}(q_0, \ell, Y_b, \neg T, q_1) \left(1 - B_{T|N}^{t-1}(\ell, Y_b, q_0)\right) B_P^{t-1}(\ell - 1, \langle a, b \rangle, q_0) \cdot$
                $B_Q^{t-1}(q_0) + \pi_2^{t-1}(q_0, \ell, Y_b, T, q_1) B_{T|N}^{t-1}(\ell, Y_b, q_0)$
                $B_P^{t-1}(\ell - 1, \langle a, b - 1 \rangle, q_0) B_Q^{t-1}(q_0)\Big] / B_Q^t(q_1)$
            $\Pr(P_{\ell-1}, \langle a, b \rangle | \mathcal{E}^{t-1}, T_\ell^{t-1})$ +=
              $\Big[\pi_2^{t-1}(q_0, \ell, Y_b, \neg T, q_1) \left(1 - B_{T|N}^{t-1}(\ell, Y_b, q_0)\right) B_P^{t-1}(\ell - 1, \langle a, b \rangle, q_0) \cdot$
                $B_Q^{t-1}(q_0) + \pi_2^{t-1}(q_0, \ell, Y_b, T, q_1) B_{T|N}^{t-1}(\ell, Y_b, q_0) \cdot$
                $B_P^{t-1}(\ell - 1, \langle a, b - 1 \rangle, q_0) B_Q^{t-1}(q_0)\Big] / B_Q^t(q_1)$
      for $b \leftarrow 1$ **to** m
        if $Y_b \in N$
          then $B_N^t(\ell, Y_b, q_1)$ += $B_P^t(\ell - 1, \langle a, b \rangle, q_1)$
            $\Pr(N_\ell = Y_b | \mathcal{E}^{t-2}, Q^{t-1} = q_1, T_\ell^{t-1})$ +=
              $\Pr(P_{\ell-1} = \langle a, b \rangle | \mathcal{E}^{t-2}, Q^{t-1} = q_1, T_\ell^{t-1})$
          else $B_\Sigma^t(Y_b, q_1)$ += $B_P^t(\ell - 1, \langle a, b \rangle, q_1)$
  COMPUTE-T($grammar, B, d, q_1, t$)

Figure 5.15: Pseudocode for prediction phase of PSDG inference algorithm without completely observable states.

allows the recognizing agent to summarize its entire sequence of observations while incurring time and space complexity costs that are only sublinear in the space of possible plan instantiations.

However, the time and space complexity is quadratic in the number of state instantiations consistent with our observations. This cost is potentially prohibitive, since the number of such state instantiations grows exponentially with the number of unobserved state variables. Section 7.1.2 discusses some methods for limiting this cost, but this complexity is clearly the limiting factor when determining the tractability of the PSDG approach in a given domain. Of course, we must first develop the PSDG specification of the domain before we consider its tractability. The next chapter analyzes the representational power of the language model.

CHAPTER 6

Modeling Domains with PSDGs

The inference algorithms of Section 5.2.3 exploit the particular independence assumptions of the PSDG definition, but the defined model is useful only in domains where those assumptions hold. As a first step toward characterizing the utility of PSDGs, Section 6.1 demonstrates the representational equivalence of the PSDG and PCFG models. Section 6.2 presents PSDG representations of two plan recognition domains, traffic monitoring and air combat. The solutions to these specific problems illustrate many of the domain features representable within the PSDG language. Section 6.3 compares the PSDG model against other recognition languages to clarify what generality it sacrifices in domain representation and what efficiency it gains in inference.

6.1 Equivalence of PSDGs and PCFGs

6.1.1 Finite State Space

It is not immediately obvious whether the state variable of the PSDG model extends the space of representable distributions beyond that covered by the original PCFG representation. In fact, for finite state spaces, we can represent every PSDG distribution with a corresponding PCFG. As a proof, this section presents a construction of a PCFG $\langle \Sigma', N', S', P' \rangle$ representing an identical probability distribution to a given PSDG $\langle \Sigma, N, S, Q, P, \pi_0, \pi_1 \rangle$. Since the PCFG cannot include any state information, we must merge the random variable Q into the space of symbols.

To maintain the context-free assumption, each new symbol must include enough state information to render its expansion independent of its ancestors and siblings. Therefore, we assign $N' = Q \times N \times Q$, so that each symbol in N' is a tuple $\langle q_i, X, q_f \rangle$, indicating

that the PSDG symbol X will be expanded starting in initial state q_i and ending in final state q_f. The PSDG expansion of X is conditionally independent of all previous expansions given the initial state q_i, and all future expansions of X's nondescendants are conditionally independent of the subtree below X given its final state q_f. We similarly assign the set of terminal symbols $\Sigma' = Q \times \Sigma \times Q$. Given these new symbol sets, we can convert a state-dependent production of the form $X \rightarrow Y_1 Y_2 \cdots Y_m$ into a set of context-free productions of the form:

$$\langle q_0, X, q_m \rangle \rightarrow \langle q_0, Y_1, q_1 \rangle \langle q_1, Y_2, q_2 \rangle \cdots \langle q_{m-1}, Y_m, q_m \rangle \ \ (p)$$

We must add such a production for every possible sequence of intermediate states, q_0,q_1,...,q_m $\in Q^{m+1}$.

To determine each production probability p, we must first specify the probability that the expansion of symbol X in initial state q_i will produce a final state of q_f, for each new symbol $\langle q_i, X, q_f \rangle$. The PSDG already defines the transition probability function $\pi_1(q_i, x, q_f)$ for terminal symbols $x \in \Sigma$. We can extend the definition of this transition probability function over nonterminal symbols as well. For a nonterminal symbol $X \in n$, we define the transition probability by considering all possible expansions of X and all possible intermediate state sequences that could provide the context for the symbols on the right-hand side. The probability of each such production and state sequence is the product of the production probability and the transition probabilities of the symbols on the right-hand side. However, for each symbol on the right-hand side, we must consider the possible initial states generated from the expansion of the previous sibling symbol (or, for the first symbol on the right-hand side, the initial state specified by the parent symbol). We must therefore sum over all possible intermediate state sequences over the expansion of the right-hand side:

$$\begin{aligned} \pi_1(q_i, X, q_f) \quad = \quad & \sum_{X \rightarrow Y_1 \cdots Y_m \ (p)} p(q_i) \cdot \\ & \sum_{q_1 \in Q} \cdots \sum_{q_{m-1} \in Q} \pi_1(q_i, Y_1, q_1) \pi_1(q_1, Y_2, q_2) \cdots \pi_1(q_{m-1}, Y_m, q_f) \end{aligned} \quad (6.1)$$

The production probabilities in the PCFG rule base P' follow a similar form, resulting in complete productions of the form:

$$\begin{aligned} \langle q_0, X, q_m \rangle \quad \rightarrow \quad & \langle q_0, Y_1, q_1 \rangle \langle q_1, Y_2, q_2 \rangle \cdots \langle q_{m-1}, Y_m, q_m \rangle \\ & \left(p(q_i) \left(\prod_{t=1}^{m} \pi_1(q_{t-1}, Y_t, q_t) \right) / \pi_1(q_0, X, q_m) \right) \end{aligned}$$

To complete the specification, we add productions that expand the PCFG start symbol S' into the possible configurations of the PSDG start symbol S, for all possible initial and final

states $q_i, q_f \in Q$:

$$S' \rightarrow \langle q_i, S, q_f \rangle \;\; (\pi_0(q_i)\pi_1(q_i, S, q_f))$$

It is straightforward to show that the probability of a given PSDG parse tree τ, with a terminal sequence of length n, is identical to the corresponding parse tree from this constructed PCFG. If we define p_{ijk} to be the probability function for the production P_{ijk} expanding the symbol at position (i, j, k) of τ, we can express the probability of that tree as:

$$\Pr(\tau) = \pi_0(Q^0) \left(\prod_{i,j,k} p_{ijk}(Q^{i-1}) \right) \left(\prod_{i=1}^{n} \pi_1(Q^{i-1}, N_{i11}, Q^i) \right) \tag{6.2}$$

The corresponding PCFG parse tree τ' begins with the production

$$S' \rightarrow \left\langle Q^0, S, Q^n \right\rangle$$

with probability $\pi_0(Q^0)\pi_1(Q^0, S, Q^n)$. Below the root node, there is a 1-1 correspondence between the symbol nodes of the two parse trees with identical (i, j, k) indices, where if $N_{ijk} = X$ in the PSDG parse tree, then $N_{ijk} = \langle Q^{i-1}, X, Q^{i+j-1} \rangle$ in the PCFG parse tree. The probability of the overall parse tree is:

$$\Pr(\tau') = \pi_0(Q^0)\pi_1(Q^0, S, Q^n) \left(\prod_{i,j,k} \Pr(P_{ijk}) \right) \tag{6.3}$$

Each probability in the product consists of a production probability $p_{ijk}(Q^{i-1})$ and a product of child transition probabilities $\pi_1(Q^{i_t-1}, Y_t, Q^{i_t+j_t-1})$, all divided by the parent transition probability $\pi_1(Q^{i-1}, X, Q^{i+j-1})$. Over all (i, j, k), the transition probability of each symbol occurs exactly once in the numerator, while the transition probability of each *non-terminal* symbol occurs exactly once in the denominator. We can thus simplify Equation 6.3 as follows:

$$\Pr(\tau') = \pi_0(Q^0) \left(\prod_{i,j,k} p_{ijk}(Q^{i-1}) \right) \left(\prod_{i=1}^{n} \pi_1(Q^{i-1}, N_{i11}, Q^i) \right) \tag{6.4}$$

Therefore, the constructed PCFG represents the same probability distribution over parse trees as the given PSDG, although, the constructed PCFG is much larger than the original PSDG. The PCFG symbol spaces N' and Σ' have a space complexity of $O(|Q|^2|N|)$ and $O(|Q|^2|\Sigma|)$, respectively. More ominously, the PCFG production space P' has a space complexity of $O(|Q|^{m+1}|P|)$, where m is the maximum production length of the PSDG.

$$
\begin{array}{lcll}
S & \rightarrow & a\,S & (p_{A1}(q_x, q_y) = \rho_A) \\
S & \rightarrow & bB & (p_{A2}(q_x, q_y) = 1 - \rho_A) \\
B & \rightarrow & b\,B & (p_{B1}(q_x, q_y) = \rho_B) \\
B & \rightarrow & C & (p_{B2}(q_x, q_y) = 1 - \rho_B) \\
C & \rightarrow & c\,C & \left(p_{C1}(q_x, q_y) = \left\{ \begin{array}{ll} 1 & \text{if } q_x > 0 \\ 0 & \text{otherwise} \end{array} \right.\right) \\
C & \rightarrow & D & (p_{C2}(q_x, q_y) = 1 - p_{C1}(q_x, q_y)) \\
D & \rightarrow & d\,D & \left(p_{D1}(q_x, q_y) = \left\{ \begin{array}{ll} 1 & \text{if } q_y > 1 \\ 0 & \text{otherwise} \end{array} \right.\right) \\
D & \rightarrow & d & (p_{D2}(q_x, q_y) = 1 - p_{D1}(q_x, q_y))
\end{array}
$$

Figure 6.1: Productions for PSDG representing a probability distribution over the language $\{a^x b^y c^x d^y, y > 0\}$.

6.1.2 Infinite State Space

If we allow the state space to be infinite, then PSDGs *can* represent distributions beyond those allowed by the PCFG model. For instance, the language $\{a^x b^y c^x d^y, y > 0\}$ cannot be represented by a context-free grammar [19], so it likewise cannot be represented by a PCFG. However, if we define the state space $Q = \mathcal{Z}^+ \times \mathcal{Z}^+$ to record the values of x and y, then we can use the productions and production probabilities of Figures 6.1, where $\rho_A, \rho_B \in (0, 1)$ are constants representing the likelihood of adding an additional a or b, respectively. In other words, the probability that this PSDG generates a string whose initial sequence of a symbols has length x is $\rho_A^x(1 - \rho_A)$, while the probability that the subsequent sequence of b symbols has length y is $\rho_B^{y-1}(1 - \rho_B)$. The PSDG is completely deterministic after the selection of production $B \rightarrow C$. The productions for C and D use the state variable $(q_x, q_y) \in Q$ as a counter recording how many c and d symbols are left to generate. We initialize the counter to be 0 by defining $\pi_0(0, 0) = 1.0$. Figure 6.2 shows the transition probability function π_1 that adjusts the counter as needed, resulting in a probability distribution where $\Pr(a^x b^y c^x d^y) = \rho_A^x(1 - \rho_A)\rho_B^{y-1}(1 - \rho_B)$.

Therefore, if we allow an infinite state space, we can represent distributions unrepresentable under the PCFG model. However, even if we can represent a distribution with a PSDG, inference is feasible only when our observations leave us with a finite number of

$\mathbf{q_i}$	x	$\mathbf{q_f}$	$\pi_1(\mathbf{q_i}, x, \mathbf{q_f})$
(q_x, q_y)	a	$(q_x + 1, q_y)$	1.0
(q_x, q_y)	b	$(q_x, q_y + 1)$	1.0
(q_x, q_y)	c	$(q_x - 1, q_y)$	1.0
(q_x, q_y)	d	$(q_x, q_y - 1)$	1.0

Figure 6.2: Transition probability function for PSDG representing the language $\{a^x b^y c^x d^y, y > 0\}$.

possible states; otherwise, we must maintain an infinite set of belief states. If we view the state variable as a vector of separate substate variables, either hidden or observed, then inference is feasible only if all of the hidden substate spaces are finite. The size of the observable substate spaces has no effect on the complexity of inference, as long as we do not need to compute a distribution over the future values of these substates.

6.1.3 Implications of Relationship between PCFG and PSDG Models

Although allowing an infinite space does extend the possible PSDG distributions beyond those representable by PCFGs, the requirement of finite belief states limits the practicality of inference with such distributions. However, even in problem domains where the PSDG representation has a finite space, a PSDG domain specification provides advantages over an equivalent PCFG. The PCFG construction algorithm presented in the proof of Section 6.1 produces a production set of size $O(|Q|^{m+1}|P|)$. The PCFG inference algorithms all have time complexity polynomial in the size of the production set, so for all nontrivial distributions, direct application of the PSDG inference algorithms, which are at most quadratic in the state space, is more efficient than construction of an equivalent PCFG and application of PCFG inference algorithms.

It is theoretically possible that domains exist where we can construct (by some other method) an equivalent PCFG for which inference is more efficient than the PSDG algorithms. However, although the equivalent PCFG may have a different production structure, its symbol structure must still represent the same set of plans and states as the original PSDG to support the same range of recognition queries. Unless the expansion of a particular plan is independent of state, we can write context-free productions only for nonterminal symbols that represent both the given plan and the relevant context. Otherwise, the probability of the production would be dependent on any omitted state variables and,

subsequently, on the other symbols in the parse tree that depend on common state variables. Therefore, the PCFG domain representation can match the efficiency of the PSDG representation only when the plan selection process has extremely limited and localized state dependency. The state transition probability distribution poses a similar problem for the PCFG representation. Of course, even in domains where the PCFG representation is more efficient, the separation between the plan and state spaces in the PSDG model can provide a more suitable modeling language, since the dependency structure more closely mirrors that of most planning domains. In such cases, we can use the PSDG model to simplify the initial specification of the domain model, and then convert the PSDG to an equivalent PCFG for inference.

6.2 Examples of PSDG Domain Representations

The PSDG domain specifications of this section illustrate the utility of the language as a modeling tool. Section 6.2.1 presents a more sophisticated PSDG representation of the traffic monitoring example, illustrating how many common plan dependencies fit within the state-dependent production structure. Section 6.2.2 demonstrates how we can specify a PSDG for an air combat domain using a previously generated specification in a different language.

6.2.1 Traffic Monitoring

State Variables for Traffic PSDG

To extend the PSDG of Figure 5.4 to a more precise model of the observed driver, we must first explicitly specify the state space, described briefly in Section 5.1. Most of the features under consideration correspond to the state of the cars on the highway. For the observed car, we record its lateral position (Xpos in units of feet), position along the length of the highway (Ypos in units of .01 miles), speed (Speed to the nearest 5 mph), and turn indicator status (Signal, either left, right, or off). We also maintain an array of features corresponding to the speeds of the cars around the observed driver. One row of the array contains the speeds for those cars to the immediate front of the observed driver, with three columns indexed by the lane positions of those cars. Another row contains the speeds for the cars immediately behind the observed driver, again indexed by lane. The third row corresponds to the cars roughly even with the observed driver. If there is no car in a given

position, the corresponding feature takes on a null value. Thus, there are at most eight other cars to consider, three rows by three lanes, but not counting the observed car itself. All of these features are observable, so we can easily expand this array and incur only negligible additional cost in inference. For clarity, we identify the elements of the array as YX-car, with Y being either F, M, or B indicating the position in front of, even with the middle of, or behind of, respectively, the observed car, and with X being either L, M, or R indicating the left, middle, or right lane.

We must also consider the unobservable preferences of the observed driver. One preference represents the driver's intended exit (Exit), the point at which it will leave the highway. We assume that the recognizing agent knows the possible exit points along the highway; otherwise, the state space would include all possible points, equivalent to the domain of Ypos. Another feature represents the driver's desired speed (DesiredSpeed), the speed (to the nearest 5 mph) at which the observed driver will travel, if not prevented by the current highway conditions. We also include an additional feature, Type, with values normal, cautious, and aggressive, indicating the driver's tolerance for the presence of other cars when making lane changes. For instance, a driver with Type=aggressive is less concerned about cutting off the cars behind it when changing lanes. The sizes of the state spaces of these unobservable features do have a significant impact on the time and space complexity of inference, so there is a serious tradeoff in the size of the domain of Type variable versus the accuracy in representing the different types of driving behaviors. The discussion of PSDG inference in this traffic monitoring example analyzes this tradeoff in more detail.

The state variable Q is simply the conjunction of all of the features described. We can specify the prior probability distribution over the state by specifying a prior probability distribution over each of the features separately, if the features are independent. The complete PSDG specification in Appendix A assumes that the prior probabilities over the features are independent of each other, although we could imagine (and easily implement) a dependency between a driver's desired speed and its Type. The prior probability distribution over the observed variables is irrelevant once we observe their initial values, since all of the query probabilities are conditioned on the initial observations.

Symbols for Traffic PSDG

The state space of the Xpos variable measures the lateral position to the nearest foot, so we can no longer use the terminal symbols of Figure 5.4, where lane changes were atomic

actions. A more useful model breaks the lane changes into the smaller changes in lateral position. With such a model, a recognizing agent could predict a lane change based on an initial movement away from the center of the lane and thus possibly avoid a collision that could take place when the lane change is completed. We introduce new terminal symbols MoveL, MoveR, and NoMove to represent the corresponding low-level actions.

We must introduce additional terminal symbols to account for the dynamics of the Speed state variable. Drivers choose acceleration maneuvers in conjunction with their lateral movements, so we could modify the terminal symbols to represent a conjunction of the two types of maneuvers (for instance, $\langle$MoveL, Accelerate$\rangle$). However, we can treat the two maneuvers as completely orthogonal, inasmuch as the set of state variables affected by the lateral movements ({Xpos}) does not overlap the set of variables affected by accelerations ({Speed}). The acceleration terminal symbols, Accelerate, Decelerate, and NoAccelerate, can now appear as separate leaf nodes. Now, the episode between successive observations must contain two terminal symbols. The first corresponds to the lateral movement, when we process the world dynamics of the affected Xpos variable. The second corresponds the acceleration component, when we process the world dynamics of the affected Speed variable.

The Signal variable requires a third component to represent the driver's use of the turn indicator. We could eliminate this third component with the atomic lane changes of the PSDG in Figure 5.4, since we could model the dynamics of the Signal variable as dependent on the driver's choice of lane change. For instance, we could include a rule stating that a driver of Type=cautious signals left with probability 0.999 when executing action Left. However, when we move to the low-level lateral movement actions, then we can no longer express the dependency of the signal on the higher-level lane change choices. Therefore, we introduce new terminal symbols, SignalL, SignalR, SignalOff, that are explicitly dependent on the choice of lane changes. Each episode between successive observations now has three terminal symbols, but we still maintain the orthogonality with respect to affected state features.

We must alter the set of nonterminal symbols to generate the separate component terminal symbols. The lateral maneuvers remain the primary focus of the planning process, so we keep the core set of nonterminals of Figure 5.4: Drive, Stay, Left, Right, Pass. We add double lane changes, 2Left and 2Right, to model the potential decision to shift two lanes (for instance, if the desired exit approaches when the driver is in the leftmost lane). The nonterminal symbol ChooseAcc can generate any of the three terminal acceleration symbols, so

in cases when the choice of acceleration is independent of the choice of lane change, we can write a production of the form Left → MoveL ChooseAcc SignalL. With such productions, we can avoid having to write expansions of Left with all of the possible acceleration choices, achieving a more modular production structure.

However, a driver is not always free in its choice of acceleration, since passing usually requires an accompanying acceleration. Two new symbols, LeftAcc and RightAcc, represent left and right lane changes, respectively, but with the additional stipulation that the acceleration symbol be Accelerate. In other words, we include productions of the form LeftAcc → MoveL Accelerate SignalL. We can then enforce the acceleration in passing maneuvers through productions of the form Pass → LeftAcc RightAcc.

In this model, the choice of acceleration maneuver is independent of choice of lane change for all but the passing maneuver. If we found this assumption of independence to be invalid, we may choose to remove the ChooseAcc subplan and express the dependency through direct productions of the form Left → MoveL Accelerate SignalL. The limited dependency does not justify creating such a set of productions for the acceleration component. However, the signaling component is clearly dependent on the choice of lane change, so it would be unreasonable to create a separate subplan ChooseSignal to be expanded independently. The production formats listed here make the choice of signaling maneuver explicit.

The shift away from atomic lane changes introduces many complications, including the two phases of the lane change: waiting for safe conditions to make the lane change, and then actually moving the car from one lane to the other. The new symbols StartLeft, StartRight, StartLeftAcc, and StartRightAcc correspond to the former phase, while the other lane change symbols, Left, Right, LeftAcc, RightAcc, correspond to the latter. The discussion of the productions makes the nature of these phases more precise.

World Dynamics for Traffic PSDG

Most of the relationships expressed by the world dynamics are straightforward, given the definitions of the state variables and terminal symbols. For instance, the value of Xpos at time $t+1$ will be to the left of its value at time t, given an interposing MoveL action. There is uncertainty in the exact change in value, as expressed by the probability distribution in the complete PSDG of Appendix A. The value of Ypos at time $t+1$ has a deterministic relationship on its previous value and the value of Speed at time t. We could introduce a dependency on the choice of acceleration, but such an alteration offers only a minimal gain

in accuracy, since we already model the Speed variable with the obvious dependency on acceleration choice, as well as the previous value of Speed.

The current model ignores any dependency between the observed driver's actions and the movements of the other cars on the highway. We could model such a dependency without a significant increase in complexity, since the movements of the other cars are observable. However, because the other cars are observable, the more accurate model would not provide any advantage in predicting their movements. The more accurate model would allow the recognizing agent to reason backwards from the movement of another car to a prediction of the observed car's low-level action. On the other hand, the observed car's position, speed, and turn indicator, all of which are observable, are much better indicators of the low-level action, so there is little advantage in using the more accurate model.

The three unobserved state variables, DesiredSpeed, Exit, and Type, represent the observed driver's preferences, which we model here as time invariant. Therefore, the values of these variables remain constant throughout the entire plan instantiation. There may be cases where this assumption is invalid, but at worst, the dynamics of these state variables depend on highway conditions and not on actions taken. For instance, we could imagine that a driver might be more (or less) willing to accept a slower driving speed after a prolonged period of heavy congestion. However, it is more likely that the driver's speed preference is unchanged, even though it accepts the slower driving speed mandated by the traffic conditions. It is difficult to imagine a scenario where the driver's maneuvering would have any effect on its preference for speed or exit point.

Top-Level Productions for Traffic PSDG

The main difficulty in domain specification lies in specifying the production structure of the PSDG. The top-level decision, among expansions of Drive, has the most complex production probability function. All of the expansions of Drive (plus the analogous expansions for new symbols 2Left and 2Right) appearing in Figure 5.4 appear in the complete PSDG, as we again model the driving process as a continuous sequence of independent episodes, terminating with an Exit.

We must then encode the observed driver's decision procedure within the production probability functions. In choosing among the possible subplans for a given episode, a driver evaluates the overall highway situation, considering all of the observable conditions, as well as considering its desired speed and intended exit point. We can view the set of probability

functions for all expansions of Drive as a decision tree, with the state features, as well as the particular expansion under consideration, as the inputs. For instance, the preference over exit points takes precedence over all other needs, so the topmost branch point of the decision tree would evaluate whether the current value of Ypos indicates proximity to Exit. If so, then the production of Drive $\rightarrow$ Exit has probability one, while all other productions have probability zero.

If the driver is not exactly at the desired exit, but only approaching very near to it, then it must move over to the rightmost lane. In such a case, it chooses a maneuver based on the value of Xpos. If the driver is already in the rightmost lane, then the production Drive $\rightarrow$ Stay Drive has probability one. If the driver is in the middle or leftmost lane, the expansion to a Right or 2Right, respectively, has probability one.

If the driver is nowhere near its desired exit, then it chooses its maneuver based on the speeds of the other cars around it. If the observed driver has no car blocking its front, it will most likely choose to stay in its current lane. The conditions for an unobstructed front require either having no car in front, or a car traveling at a speed faster than the driver's desired speed. We can establish whether these conditions hold by examining the state variable corresponding to the speed of whatever car is in front of the observed driver, as well as DesiredSpeed. If the driver's front is unobstructed, then the production Drive $\rightarrow$ Stay Drive has a high probability.

Otherwise, the observed driver first determines which lane will best support its desired speed, by comparing the average speeds of the cars it sees in the other lanes, using the array of car speed variables. The driver attempts to move into the lane where the average traveling speed is closest to DesiredSpeed, favoring lanes to the right when breaking ties, so the production probability function has a high probability for the maneuver that moves the driver into that lane. If the observed driver's current lane has the most suitable traveling speed, then the production probability function has a high probability for the expansion Drive $\rightarrow$ Pass Drive, since the observed driver will most likely pass the car obstructing its front if it wishes to stay in its current lane and still reach its desired speed.

Passing Productions for Traffic PSDG

A driver executing a passing maneuver must first wait for the other cars to give it room to execute its pass (Pass $\rightarrow$ Stay Pass), at which point it has two possible methods for passing (Pass $\rightarrow$ StartLeftAcc StartRightAcc, or Pass $\rightarrow$ StartRightAcc StartLeftAcc). However, in

the process of waiting for room to pass, the driver may find itself approaching its intended exit. If it does, it will abort its passing maneuver and prepare to exit. We model this possible premature termination with a production Pass $\rightarrow$ Stay that terminates the passing maneuver, whereupon the driver will expand the subsequent Drive according to its exiting requirements.

We again define the production probability function as a decision tree, using all of the features of the state space as inputs. If the value of Ypos is close enough to that of Exit, the termination production has probability one, while all other productions have probability zero. The termination production also has probability one if the front of the car is no longer obstructed, since there is no need for the observed driver to pass. Under any other state configuration, the termination production has probability zero. We eliminate impossible maneuvers, like passing on the left from the leftmost lane, by defining the production probability function to be zero for the corresponding productions under the appropriate conditions.

In all other situations, the driver evaluates the possibility of passing on either the left or right. If another car is present to its left (right), then passing on the left (right), has low probability. The observed driver also considers the presence of any cars ahead of it and to its left (right). If another car is present at that position, a driver of type cautious will not pass to the left (right), but drivers with a type of either normal or aggressive will consider passing on the left (right) as long as the car present is traveling faster than their own current speed. The observed driver likewise considers cars behind it and to its left (right). Again, a driver of type cautious will not pass to the left (right) if another car is at that position. A driver of type normal will consider passing to the left (right) as long as the interfering car is traveling slower than Speed. A driver of type aggressive does not consider the presence of cars behind it when deciding whether to pass.

If these criteria do not eliminate passing on either left or right, the production probability function has a high probability for passing on the left, though the exact value depends on the driver's Type. If the passing criteria eliminate passing on the right but not the left, passing on the left has an even higher probability. If the criteria eliminate passing on the left but not the right, then passing on the left has a very low probability, but passing on the right may still not have a very high probability. A driver with Type=cautious stays in its current lane with probability 0.5, waiting for a future opportunity to pass on the left. On the other hand, a driver with Type=aggressive is largely indifferent between passing on

the left and on the right, so passing on the right will have a high probability. If the criteria eliminate both passing maneuvers, then the driver chooses to wait (Pass → Stay Pass) with high probability, again depending on the value of Type.

Lane Change Productions for Traffic PSDG

The four lane changes (left, right, left with acceleration, right with acceleration) have analogous productions, so this section presents the standard left lane change as the example. The productions for the other lane changes have the obvious differences, with the lane changes with acceleration replacing occurrences of ChooseAcc with Accelerate as required. The left lane change always begins with StartLeft, representing the driver's waiting for conditions to allow a safe lane change. As with the passing maneuver, we use a termination production (StartLeft → Stay), which has probability one if the driver's intended exit approaches.

The observed driver's choice between waiting (StartLeft → Stay StartLeft) and beginning its left lane change depends on the same criteria used in evaluating the safety of passing on the left, again considering the speeds of the cars in the lane to its left, as well as its own speed and aggressiveness. If the state does not meet these criteria, the production probability function has a high probability for the waiting expansion. If the state *does* meet the criteria, then the function has a high probability for the expansion StartLeft → MoveL ChooseAcc SignalL Left. It also has a smaller probability for the unsignaled variant of this production (SignalOff instead of SignalL).

The Left subplan has two possible expansions, ignoring the signaling component for now. During the lane change, the driver chooses the production Left → MoveL ChooseAcc SignalL Left. Once Xpos reaches the middle of the target lane, the lane change maneuver terminates with the production Left → Stay. Dividing the lane change into two subplans, StartLeft and Left, permits a simple definition of lane change termination where the production probability function does not need to know the specific target lane. Instead, the StartLeft subplan initiates the Left subplan only after performing a MoveL to first move away from the center of the current lane. The production probability function for the expansions of Left choose the termination production only when Xpos coincides with the center of *any* lane. Of course, under this termination condition, the driver must be at the center of the target lane, since moving left from its current lane moves it into the center of the target lane before that of any other lanes.

As with the expansions of StartLeft, we must consider the possible signaling maneuvers. If Signal=left, then the expansion with SignalL has a much higher probability than that with SignalOff or SignalR, since it is unlikely that a driver would change from the correct turn indicator position. If Signal=off, then there is a high probability for the expansion with SignalOff, but the expansion with SignalL has a significant probability, since a driver may correct its signal in the middle of the lane change. If Signal=right, the expansion with SignalR has the highest probability, since the driver is unlikely to notice such a mistake.

In the current model, Stay has only one possible expansion: Stay $\rightarrow$ NoMove ChooseAcc SignalOff. This model does not account for a driver's signaling until it actually initiates its lane change. We could model a driver's anticipatory signaling by introducing new nonterminal symbols StayR and StayL whose expansions replace SignalOff with SignalR and SignalL, respectively. We could then replace the Stay subplans of the lane change expansions with these signaling variants, with a probability representing the driver's likelihood of signaling prior to initiating its lane change.

Acceleration Productions for Traffic PSDG

There are three possible expansions for ChooseAcc: ChooseAcc $\rightarrow$ X, where X can be Accelerate, Decelerate, or NoAccelerate. The production probability function weighs the relationship of the driver's current speed to its desired speed, as well as the speeds of the cars to its immediate front and back. If its front is obstructed by a car traveling slower than Speed, the driver chooses Decelerate to avoid a collision. If the driver does not have to worry about such a collision and if its Speed<DesiredSpeed, it chooses Accelerate with high probability. If Speed> DesiredSpeed (as is possible after a passing maneuver), the driver chooses Decelerate, as long as any car behind it is traveling slower than Speed. Otherwise, the observed driver maintains its current speed to avoid a collision. The driver also chooses NoAccelerate when it is at its desired speed.

Inference with Traffic PSDG

Once we have the complete domain specification of Appendix A, we can use the algorithms of Section 5.2.3 to answer queries. For instance, suppose we observe a driver traveling at 65 mph in the middle lane and approaching a car traveling at only 60 mph, with no other cars nearby. We would enter the appropriate evidence for Q^0, with Xpos=10, Ypos=0.00, Speed=65, FM-car=60, and YX-pos=nil for the other positions. The PSDG recognition

system determines the probability of this evidence as $6.06 \times 10^{-}4$ according to the prior probability function. We can now query the system about the unobserved variables. We find that the probability distribution over Exit is identical to the prior uniform distribution, because the intended exit does not depend on any of the other state variables. However, the observed Speed raises the probability that the observed driver's speed is 65 mph from the initial 0.15 to 0.75.

The system also computes a posterior probability distribution over possible plans at time 1 using the plan prediction procedure. For instance, if we want to know the probability that the driver has decided to stay in its current lane, we can examine the probability that the production node P_1^1 takes on the value Drive $\rightarrow$ Stay Drive, for which the system returns 0.15. The Stay production is unlikely because of the low likelihood that the observed driver's desired speed is below the 60 mph traveling speed of the car in front of it. In this case, the production Drive $\rightarrow$ Right Drive has the highest probability at 0.75, because the driver sees no cars in the right lane and chooses it as the optimal lane location. The Pass production has probability 0.1 in this case, and the system also provides the distribution over production node P_2^1, indicating that a pass on the left has probability 0.92 given that a pass takes place (probability 0.092 overall).

A recognizing driver's more immediate concern is the distribution over any imminent moves the driver may make, and we can query the terminal symbol node Σ^1 to find that the probability that the observed car is performing a MoveR is 0.747, while the probability of a MoveL is 0.089. Thus, the probability that the car does not move in the current time slice 0.164, greater than the 0.15 probability of Drive $\rightarrow$ Stay Drive because even if the driver intends to change lanes, it may not begin executing the change immediately.

The system provides similar probability distributions over the acceleration and signal components of the planning process. The entire prediction phase over the three components takes 3–4 seconds on a SUN Sparc machine. A recognizing driver is not likely to have that much time between successive observations, but Section 7.1.2 proposes a method for reducing that time to less than a second.

Once the recognizing agent does receive its next set of observations, it provides the system with the corresponding values for Q^1, initiating the explanation phase. For instance, suppose we observe our driver move to the right (Xpos=8) while accelerating (Speed=70). The other state variables remain unchanged, except that the driver has moved a little further down the highway, so that Ypos=0.03. The system computes new probabilities over

the plan symbols at time 1, beginning with the terminal symbols. The posterior probability distribution over Σ^1 now indicates with certainty that the observed driver has performed a MoveR. As a result, the system determines that the Stay subplan has zero probability. The probability of Drive $\rightarrow$ Right Drive increases to 0.92, while the probability of a passing maneuver drops to 0.08. The high likelihood of passing on the left versus passing on the right justifies the lower degree of belief in the passing explanation.

Having completed the explanation phase, the system commences to predict possible plan states for time 2. In this case, the observed driver has just initiated a right lane change, so it will continue to move to its right until it reaches the right lane. The system therefore carries the explanation probabilities over plans at time 1 over to time 2, because none of those plans will have terminated. It then determines that the driver will perform another MoveR at time 2 with probability one, because both the single lane change and passing explanations require moving to the right.

The system also computes a posterior distribution over the values of the unobserved variables. The observed driver's rejection of the Stay plan increases our belief that DesiredSpeed¿60. For instance, the system determines that $\Pr(DesiredSpeed = 70|\mathcal{E}^1) = 0.21$. In addition, the driver began its lane change immediately, slightly lowering our belief that Type=cautious to 0.088. The current Ypos is not near any exit, so our belief over Exit remains unchanged. If we were to observe this same scenario at some later time when the driver *is* near the first exit (Ypos=3.5), then our belief that Exit=4 would increase to 0.73, while our belief in the passing explanation would decrease further to 0.046.

We now jump ahead in our original example from time 1 to time 4, when the observed driver has completed its right lane change with Xpos=3, Ypos=0.14, and Speed=70. The YX-car variables are all null, except for MM-car, which is 60 mph, representing the car that was initially obstructing the observed car. The system's prediction phase considers two possibilities: the driver has completed a single lane change and is choosing a new maneuver, or it has to complete the passing maneuver chosen at time 1. In the former case, the driver chooses to the Stay subplan with probability one, because there are no cars in its path. However, it may choose to accelerate if DesiredSpeed$>$ 70, but after three successive observations of Speed=70, the system determines the probability of such a state to be 0.03. If the driver is in the midst of a passing maneuver, it will most likely wait to be clear of the car being passed before moving back to the middle lane. Therefore, the overall probability of a NoMove action is 0.976, with MoveL the only alternative. However, a recognizing driver

would believe that an eventual return to the middle lane will occur with probability 0.073, even though immediate movement is more unlikely.

The system computes probabilities over the low-level actions, while still maintaining a distribution over the more complex plans and intermediate plan states. The former beliefs form the basis for more immediate reactions, like recognizing that the observed driver in the example may choose a MoveL action at time 5 with probability 0.024 and cut off the car being passed. The beliefs over higher-level plans support long-range strategizing. For instance, consider the case of a third car approaching the observed driver in the right lane at a speed greater than 70 mph. The driver of this third car may compute the high probability of the Stay subplan at time 5 and decide to shift to the leftmost lane to avoid both of the other cars, even though it is possible that the observed car will return to the middle lane and leave the rightmost lane clear. The PSDG representation of the traffic domain thus supports many of the queries desired by driving agents in their interactions with other cars on the highway.

6.2.2 Air Combat

The traffic monitoring example demonstrates how we can generate a PSDG domain specification from scratch. In many domains, we may have a pre-existing domain specifications in some other language. For instance, the air combat domain has provided a fertile problem area for plan recognition research [42], since a pilot making decisions about its own actions must consider the possible goals and actions of an enemy. An observed pilot's decision process combines goal-driven and reactive behaviors in fulfillment of overall mission goals, and the recognizing pilot must make its decisions within rigid time constraints, so the problem demonstrates many of the key issues that arise in more typical plan recognition problems.

The pilot's top-level goal, Execute-mission, appears at the top of the operator hierarchy representing a subset of air combat maneuvers. The execution of a particular plan requires the execution of some combination of its child plans. The choice of the exact combination depends largely on the world state, which covers the positions and speeds of any planes and missiles in the area, as well as information about the capabilities (weapons, radar, etc.) of each plane. For instance, a pilot can execute an Evade plan to avoid an oncoming missile by choosing either Beam-left or Beam-right, depending on the relative position of the plane and the missile. These subplans ultimately invoke some combination of the low-level turning

actions.

Existing Air Combat Domain Specification

The agent tracking research to this domain represents these plan relationships with productions in the Soar architecture citeNewell to perform recognition. For instance, the following Soar production proposes a Get-steering-circle subplan in the course of executing a Launch-missile subplan:

```
(sp launch-missile*suggest-proposal*get-steering-circle*bogey
   (goal <g>  ^state <ts>)
   (<ts> ^operator-stack <lm>)
   (<lm> ^agent <team> ^actual launch-missile ^bogey <me> ^last <team>)
   (<team> ^type team ^paradigmatic <b>)
   (<b> ^state <s> ^name bandit)
   (<s> ^selected-missile <m> ^missile-info <mi>)
   (<mi> ^object <mo>)
   (<mo> ^type <m> ^lar-achieved <me> -^steering-circle-achieved <me>)
 -{ (<ts> ^operator-stack <mm>)
    (<mm> ^actual model-missile ^agent.paradigmatic <b> ^bogey <me>
          ^missile-info <mo>)
  }

→

   (<g> ^operator <o> +)
   (<o> ^name <gsc>)
   (<gsc> ^actual get-steering-circle ^missile-info <mo> ^bogey <me>
         ^agent <team> + & ^child-of <lm>))
```

The first line (following `sp`) is simply the name of the production. The next few lines preceding → are the conditions of the production, with free variables represented in the form <x>. Each condition begins with an object identifier, possibly a previously bound variable, and is followed by slot names (preceded by the caret). The conditions specify either a particular value to match against the slot contents, or a free variable to be bound to the value of the slot contents. For instance, the first condition looks for a `goal` construct (subsequently bound to free variable `<g>`) and binds free variable `<ts>` to the contents of the

`state` slot. This first condition is true for any goal `<g>` that has a `state` slot. Productions include such conditions to access nested slots of interest. The condition on `<mo>` is more interesting because it requires that the `lar-achieved` slot contain `<me>`, indicating that the observed pilot has reached the launch acceptability region (LAR) with respect to its opponent (bound from the `bogey` slot in the `<lm>` condition). This condition also requires that the `steering-circle-achieved` slot *not* contain `<me>`, where the `-` preceding the slot name indicates negation. Likewise, the `-` preceding the last two conditions (within the braces) indicates the negation of those conditions.

The lines after → are the results of firing this production, in the form of actions specifying the creation, modification, or deletion of objects. The first line alters the previous bound `<g>` construct to contain a new `operator` slot filler `<o>`. The second line specifies that this new filler have new object `<gsc>` as its `name` slot filler. The `<gsc>` object represents the details of the new operator of type Get-steering-circle (indicated by the `actual` slot filler), aimed at enemy `<me>`. There are additional details to this production, most notably involving teams, but we can summarize this production as a proposal of the Get-steering-circle subplan, conditional on an existing Launch-missile subplan where the observed pilot has achieved LAR, but not steering circle, with respect to its enemy.

PSDG Generation from Soar Productions

There are obvious similarities between the purpose of this Soar production and the PSDG production definitions. However, the PSDG model restricts productions more than the Soar production format. For instance, the above Soar production could include conditions involving ancestor subplans further up than Get-steering-circle's immediate parent Launch-missile. In addition, there are no restrictions on the number or type of objects referred to in the action portion of the production. In other words, any Soar production can introduce multiple, simultaneous subplans, remove existing subplans, or otherwise modify existing state within a single production.

Fortunately, most of the Soar productions in the air combat domain specification do not include such conditions or actions. We can therefore translate such productions into PSDG productions. We first create state variables to represent those production conditions not included in the plan hierarchy. For instance, the example production refers to conditions `lar-achieved` and `steering-circle-achieved`. These slots become state variables in the PSDG representation. We must also include lower level state variables to represent the

relative position of the observed pilot and its enemy, because the definitions of LAR and steering circle depend on that position and the observed pilot's low-level flying actions affect that position. We can thus specify the dynamics of flying actions in terms of these low-level position variables. The values of the LAR and steering circle variables are then completely determined given the position.

Once we have elicited the state variables, we generate the plan selection rules. The example Soar production corresponds to a PSDG production of the form Launch-missile $\rightarrow$ ξ_0 Get-steering-circle ξ_1. The ξ_0 and ξ_1 strings correspond to the other possible children of Launch-missile. We must generate PSDG productions for all of the possible sequences of these children. Although the Soar productions in the air combat domain specification do not provide an explicit ordering on the children, the conditions on each production do implicitly serialize much of the execution. For instance, the Soar production generating the operator for Lock-missile, another child of Launch-missile, requires the condition `steering-circle-achieved`. Since only the operator Get-steering-circle achieves this condition, Lock-missile must follow Get-steering-circle in the expansion of Launch-missile. We can thus generate the possible sequences of children necessary for the PSDG productions by examining the pre-conditions of each child and determining what siblings establish those pre-conditions.

Many Soar productions specify the removal or termination of an existing operator upon achieving certain conditions. We could, in theory, represent such a production by a PSDG expansion of the corresponding nonterminal symbol into the empty string. Unfortunately, the current PSDG definition does not allow such productions. It is unlikely that any simple modification to the PSDG inference algorithms could handle such productions, since the existing algorithms rely on time slices spanning a single terminal symbol. Such time slices could include an arbitrary number of expansions into the empty string, making it difficult to maintain a compact representation of all possible productions. However, Section 7.1.1 proposes a possible solution for such productions.

Comparison of PSDG and Original Domain Specification

Appendix B contains the PSDG specification of the air combat domain. It considers the restricted case when there is a single observed pilot interacting with at most one other enemy plane. Therefore, the conditions on `agent`, `bogey`, and other agent-identification slots are irrelevant to the PSDG specification. We could extend the PSDG representation to handle

multiple agents by including additional sets of state variables for each such agent, although we could only do so for a fixed number of agents. In the traffic example, the YX-car variables represented other possible cars on the highway. We could introduce analogous variables to the air combat PSDG, although each pilot requires a more complex representation than the single variable used to model the cars.

The Soar specification also supports the reflexive reasoning necessary when the observed pilot is trying to analyze the behavior of the recognizing pilot as well. Thus, in the agent tracking approach, the observed pilot models its enemy as performing the same planning process as itself, and the enemy pilot models the observed pilot as modeling the enemy pilot as performing the same planning process as itself, etc. Such a relationship is beyond the world dynamics representable within the PSDG language. The PSDG domain specification models the observed pilot's enemy with a static probabilistic behavior model, essentially choosing actions from a fixed distribution. Section 7.1.3 discusses possible methods for relaxing the assumption of a fixed distribution, but achieving true reflexivity is beyond the scope of this dissertation.

Within this restricted one-on-one air combat domain, the PSDG does provide a probabilistic extension beyond the deterministic reasoning of the Soar representation. We can thus account for possible deviations from the behaviors included within the Soar domain specification by introducing additional PSDG productions that have low probability. However, it is difficult to gauge the utility of this probabilistic extension because there is no basis for choosing the production probabilities within the original Soar domain specification. The PSDG productions taken directly from the original specification have an arbitrarily chosen high probability, while the deviation productions have an arbitrarily chosen low probability. Thus, the PSDG representation will not provide any advantage over the original specification in the expected case. However, if we extended the modeling effort invested in developing the Soar domain specification to assess more accurate probabilities over the plan hierarchy, a PSDG specification could provide an advantage when considering unlikely explanations that require responses beyond the expected case.

6.3 The PSDG Formalism in Relation to Other Representational Languages

These two examples of domain specification demonstrate the utility of the PSDG representation within these specific instances. Both problem domains illustrate many of the dependencies present in most plan recognition problems. However, there are significant limitations to the representational power of the PSDG model, as demonstrated by the occasional simplifying assumption employed in the domain specification process. This section compares the PSDG language against existing languages, including some not expressly designed for plan recognition, in an effort to more precisely categorize the strengths and weaknesses of the proposed formalism.

6.3.1 Event Hierarchies

If we ignore the stochastic component, the rules allowed under the PSDG model form a subset of the rules allowed under the event hierarchy formalism [26]. The symbols of the grammar, $N \cup \Sigma$, become the action types classifying event instances. As in the CFG translation of event hierarchies [43], a production of the form $X \rightarrow Y_1 \cdots Y_m$ corresponds to the decomposition rule:

$$\forall e.\text{instance}(e, X) \quad \Rightarrow \quad \text{instance}(S(1, e), Y_1) \wedge \cdots \wedge \text{instance}(S(m, e), Y_m) \wedge \kappa(e)$$

where $S(t, e)$ names subaction t of event instance e, and $\kappa(e)$ expresses constraints on the decomposition.

Although the PSDG production format can capture the relationship between X and its child subplans, the decomposition rule of the event hierarchy language has significantly more representational power. For instance, in the general event hierarchy framework, the order of the subactions does not necessarily correspond to a temporal ordering. However, the grammatical productions *do* indicate a temporal ordering, so $\kappa(e)$ must state that the time interval representing the duration of $S(t, e)$ must end exactly when the time interval for $S(t + 1, e)$ begins. Therefore, a PSDG cannot explicitly represent partial orders, but must instead create productions for each possible total order consistent with the partial order. In addition, the $\kappa(e)$ constraint expression can include conditions on any aspect of the knowledge base. The PSDG productions are conditioned on only the current state, so that they cannot represent constraints on past or future states, nor on other subplans (beyond specifying the decomposition itself).

However, inference with an event hierarchy of this general form incurs the high complexity cost of deductive inference. The PSDG inference algorithms exploit the restricted language to attain more efficient inference. Other plan recognition approaches that use representations in first-order logic make similar restrictions to allowable dependencies. Such approaches can often achieve levels of efficiency comparable to the PSDG model.

6.3.2 Plan Recognition Bayesian Networks

However, the PSDG model does provide the obvious advantage (discussed in Section 2.1.3 of a probabilistic model beyond the first-order logic of these other plan recognition formalisms. The language of plan recognition Bayesian networks (PRNs) [9] is also more general than that of the PSDG model while still supporting probabilistic inference. However, to perform inference within the PRN language, we must generate a Bayesian network representing all of the observed evidence variables, as well as all of the related unobserved plan and state variables. Once we have this network, we can answer queries about all of these unobserved variables, similar to those answerable with the PSDG Bayesian network inference algorithms of Sections 5.2.1 and 5.2.2, but beyond those answerable with the belief state manipulation algorithms of Section 5.2.3.

Unfortunately, as was the case with the static PSDG networks, the PRN representation of a PSDG will be too complex unless we restrict ourselves to either simple grammars or short periods of execution. Otherwise, the PRN must explicitly represent an impractical number of variables over the entire run. In addition, as was the case with even the DBN representation, the PSDG model does not have the properties of conditional independence that the Bayesian network inference algorithms can easily exploit.

6.3.3 Grammatical Models

The History-Based Grammar (HBG) [3] provides a rich model of context sensitivity by conditioning the production probabilities on (potentially) the entire parse tree available at the current expansion point. If we include the state variable within the grammar itself, then PSDGs are a special case of the general equivalent classes the HBG representation uses in establishing the dependency structure. However, as is the case with most grammatical representations aimed at natural language problems, the existing HBG algorithms compute only parse tree probabilities. Without a more efficient means for answering the common classes of recognition queries, grammatical representations like HBGs, PEARL [32],

and probabilistic parse tables [5], although more general than the PSDG model, remain unsuitable for plan recognition.

6.3.4 Stochastic Programs

Koller et al. introduced a new language, *stochastic programs* [28], for modeling stochastic processes. Their language is also more general than the PSDG formalism. The stochastic programming language extends a general purpose functional programming language to include an explicit probabilistic component, providing a powerful means of representing a broad range of stochastic processes. The basis for the probabilistic component is the expression **flip**(α), returning value **true** with probability α and **false** with probability $1 - \alpha$. Koller et al. demonstrate how one can model any PCFG with a stochastic program by writing a separate procedure for each nonterminal that uses the **flip** construct to choose among its possible productions. For the given production, we execute the corresponding procedure for any nonterminal symbol appearing on the right-hand side and output any terminal symbols that appear.

A stochastic program representation of a PSDG could introduce state variables, using the **flip** construct to represent the prior probability distribution π_0. We could then modify the expansion selection of the PCFG stochastic program to branch on the values of the state variables, although it is not clear how to represent the effect of the world dynamics on the state variables within the PCFG program structure. However, if we could represent the world dynamics correctly, the stochastic program would represent the same probability distribution as the original PSDG and the inference algorithms for stochastic programs can answer a broad class of queries about aspects of the program specification. However, it is not clear that the inference algorithms for stochastic programs would exploit the independence properties specific to the PSDG language.

6.3.5 Evaluation of Representational and Inferential Power of PSDG Language Model

An examination of the PSDG model with respect to the plan recognition requirements cataloged in Section 2.2.7 illustrates its specific strengths and weaknesses. For modeling the planning agent's environment, the PSDG state variable specification supports arbitrarily complex probabilistic dependency structures. We can thus specify any joint distribution over possible state instantiations, providing great flexibility in modeling the external world.

Of course, increasing the complexity of the dependency structure among the unobserved variables greatly increases the complexity of inference, so there are practical limitations to modeling the environment. The propositional nature of the state representation introduces a theoretical limitation as well. The fixed set of state variables prevents us from specifying first-order models of the world (e.g., the other cars on the highway).

The same PSDG state variable structure represents the planning agent's mental state. Again, we can specify arbitrarily complex probability distributions over the joint space of possible mental state propositions. We can thus capture probabilistic sensor models to capture uncertain noise present in agent formation. However, the finite state variable representation is insufficient for representing an infinite space of possible utility functions. If we can model the planning agent's preferences by a finite set of goals (e.g., the driver's target speed, intended exit), then the PSDG state variable representation is again sufficient. In addition, we may be able to discretize the space of utility functions, as in the Type variable of the driver model. As long as there is a finite set of utility function classes sufficient for distinguishing relevant behaviors, then we can again use the PSDG state variables to model the required preference structure.

The PSDG production structure easily represents hierarchies of plans and subplans, through abstraction and decomposition productions. The production probabilities summarize the plan selection process by specifying a probability distribution over plan choices, conditioned on the context. This production format is clearly suited for modeling agents that plan according to simple condition-action rules. However, even if the agent performs a complex decision-theoretic analysis in making its choice, we can still summarize this process through PSDG production probabilities. For each possible state, we could simulate the decision-theoretic analysis off-line to obtain the probability of a given production.

The PSDG can also represent subplan preconditions by specifying a probability zero for productions selecting a particular subplan in a context that does not meet its preconditions. The PSDG cannot explicitly specify termination conditions, although it can specify such conditions implicitly through the relative probabilities of productions continuing a subplan versus those moving on to the next subplan. For instance, in the traffic example, the productions for Left offer the choice of staying in the current lane and halting the maneuver, or moving a small distance to the left and continuing the Left maneuver. The Left maneuver is complete when the car is fully within its new lane. We can model this completion by specifying a probability of one for the production Left $\rightarrow$ Stay when the termination

condition holds. Similar production structures can represent an agent's willingness to abort a plan under certain conditions, even if the plan is not complete.

Although the conditional production probabilities allow great flexibility, the production structure itself does place a serious restriction on the representational power. Grammatical productions require a total order over the subplan steps in a decomposition. The PSDG cannot explicitly represent partial orders, nor can it represent concurrent actions (beyond orthogonal component actions like lane changes and acceleration maneuvers).

The PSDG state transition probabilities easily model the dynamics of the external world state. The conditional probabilities are sufficient to represent any joint distribution over future world states, conditioned on the past state and the low-level action taken. However, the state transition probabilities cannot represent the effects of subplan choices on future states.

The PSDG inference algorithms support a broad class of interesting queries. The Bayesian network representations support the computation of arbitrary posterior probabilities, expressed in terms of the network variables. For prediction, we can compute the probability of plans, actions, and states given arbitrary collections of evidence about other plans, actions, and states. Unfortunately, network inference is impractical in most cases. The specialized PSDG inference algorithms can compute posterior probabilities of plans and states, given observations of past states. These algorithms can also compute explanation probabilities over plans chosen before our latest observations of the state. However, the algorithms assume observations of only the state variables; they cannot exploit direct evidence about plans.

The comparison of the PSDG language models against existing models emphasizes the limited representational power of the proposed formalism with respect to these other languages. However, the PSDG language model exploits its specialized independence assumptions to support probabilistic inference in problem domains where other languages would not. Thus, the PSDG representation exchanges generality for efficiency of inference, while still meeting many of the specification requirements of interesting problem domains.

CHAPTER 7

Conclusion

7.1 Extensions to PSDG Model

The limitations of the PSDG language and inference algorithms suggest several areas for improvement. It should be possible to address some of these areas through minor modifications to the model that do not significantly increase the complexity the inference algorithms. Other areas may require a greatly altered, or possibly completely new, model. This section presents possible extensions to the original PSDG model that address many of the issues raised by the discussion of the previous chapter.

7.1.1 Generality of PSDG Representation

Most of the limitations of the PSDG model in relation to existing languages involve the restricted nature of the representation. For instance, the Soar productions for the air combat domain often specify actions terminating a particular operator. To accommodate such productions, we could augment the PSDG language to include state-dependent termination rules. For example, we could introduce the production Get-steering-circle $\rightarrow \varepsilon$ with probability one if steering-circle-achieved is true and zero otherwise. We could not consider such a production when first expanding Get-steering-circle because it would produce no terminal symbols, thus disrupting our division of time slices. However, if we first select a recursive production of the form Get-steering-circle $\rightarrow \xi$ Get-steering-circle, then upon termination of the expansion of ξ, we can test the value of steering-circle-achieved. If true, the expansion of Get-steering-circle terminates as well; otherwise, we choose a new expansion according to the original PSDG model.

This new production type alters the probabilities for the termination nodes T_ℓ^t. In the

original formalism, the values of these nodes was completely determined given the productions. This new extension would require the termination nodes to depend on the relevant state variables. However, because termination node T_ℓ^t would depend on the subsequent state variable Q^{t+1} (we terminate only if the current action achieves the desired conditions), we still have the weak form of interslice independence on which the inference algorithms rely. Therefore, a modified version of the existing PSDG inference algorithms should be able to handle the extension without incurring much additional complexity.

It would also be useful to relax the restriction that the state transition probability depend only on the terminal symbols. In the traffic example of Section 6.2.1, we introduced the signal component of the low-level action because we could not represent the appropriate dependency of the state of the turn indicator on the higher-level lane change subplans. It is trivial to alter the specification of the transition probabilities to allow such dependencies.

Unfortunately, when the states can depend on arbitrary nonterminal symbols, we can no longer localize their dependency to the terminal symbol node Σ^t. The resulting dependency of the state variables on other levels of the hierarchy can possibly introduce additional dependencies among the symbol nodes at these levels of the hierarchy when we no longer explicitly represent our observations for those state variables. Within the existing PSDG model, we need to consider the dependency only between parent-child pairs, providing the basis for the compact representation of the belief state. It is unlikely that we could add nonterminals to the state transition probabilities without requiring the belief state to include probabilities over the entire joint plan space.

One severe limitation of the production format is the restriction of serial execution. It is difficult to represent partial orders or concurrency among subplans. Of course, we can introduce multiple productions to represent all possible total orders, or treat consecutive terminal symbols as concurrent (as in the lateral movement, acceleration, and signal components in the traffic example). The former is clearly inefficient, while the latter is possible only when the components affect distinct sets of state variables.

The PSDG model would require significant modifications to handle such extensions more directly. However, ignoring the representational difficulty for the moment, the general framework of the existing PSDG algorithms could, in principle, support inference with such a representation language. The belief state as currently specified represents the probability distribution over symbols and productions within a current time slice. The current set of random variables group the symbols by level in the hierarchy because the lack of concurrency

ensures mutual exclusivity among symbols at the same level. However, we could easily reconfigure the belief state to represent symbol and production probabilities with separate variables for each symbol, where symbols at the same level of the hierarchy could now exist concurrently. The loss of mutual exclusivity would undoubtedly create a more complicated dependency structure, possibly making inference intractable, but the PSDG belief state formulation should support an exploration of these more general action configurations.

7.1.2 Complexity of PSDG Inference

In some problem domains, the PSDG inference algorithms may be intractable. The main difficulty arises from the inference algorithms' maintenance of separate belief states for each possible point in the joint space of the unobserved variables. In the example domains of Section 6.2, this joint space was small enough to support efficient inference. However, in domains with more complex models of the observed agent's mental state, usually unobserved, the complexity of inference could be prohibitive.

On the other hand, the PSDG algorithms are very efficient with respect to the production space, so we can tolerate complexity in the production structure more easily than in the space of unobserved states. As a result, we can often reduce the cost of inference by translating uncertainty about states into uncertainty about productions. For instance, in the traffic example, we can eliminate the Exit state variable by altering the expansions of Drive. Instead of explicitly considering whether the observed driver is near its intended exit, the production probability considers whether it is near *any* exit. If so, the driver chooses to prepare to exit here with a probability based on a uniform distribution over the remaining exits. In other words, if Ypos indicates that the driver is near the first exit (at mile 4), then there is a 0.2 probability that the driver will begin exiting, which will require a new nonterminal symbol PrepareExit to move the driver into the rightmost lane and exit when reaching the ramp.

This modification does not change the underlying probability distribution, since the driver's intended exit affects its decisions only when the intended exit approaches. Likewise, the driver's actions do not affect the choice of its intended action. We can still compute the same distribution over the driver's intended exit by computing the distribution over the symbol PrepareExit. If the probability is zero, then we are not near any exits, so we assume a uniform distribution over the remaining exits. Otherwise, the probability of PrepareExit is the probability that the next exit is the driver's intended exit, with a uniform distribution

over the other remaining exits.

However, we cannot use this approach to eliminate the Type and DesiredSpeed state variables. Both of these variables affect the driver's choices at many stages of the planning process and in different ways at each stage, unlike Exit, which has a fixed effect at a single point in the decision process. To eliminate the Type variable, while still maintaining coherently normal, cautious, or aggressive behavior, we have to introduce separate plan variables for each of these possible Type values. Thus, there would be symbols like AggressiveLeft, representing the behavior of an aggressive driver when performing a left lane change. The resulting growth in the symbol and production space offsets any savings from reducing the space of unobserved states.

Another potential solution is to maintain belief states for all possible values of *individual* state variables. In the traffic example, we would have three separate belief states indexed by Type, five additional belief states indexed by Exit, and ten belief states indexed by DesiredSpeed. The inference algorithms could combine the belief states for a particular point in the joint space by assuming that the separate state variables are independent. The resulting algorithms would require exponentially less time and space than the exact inference algorithms. However, because these separate state variables are actually dependent, the query responses would be only an approximation of the real values. Research in approximation in the modeling of stochastic processes shows that the error incurred by such approximations is bounded [4], but the size of the error is highly domain-dependent, so more work is needed to better understand the possible utility of such an approximation.

As an alternative, we can restrict the generality of the world dynamics and other state interactions. Although the PSDG model highly constrains the plan symbol structure to the production format, the production and state transition probabilities are arbitrary functions. We may be able to achieve more efficient inference over a restricted subset of PSDGs. For instance, the unobserved state variables in the traffic PSDG are time-invariant, so their values do not change at each new time slice. In such cases, we can reduce the time complexity of the inference algorithms to be linear in the joint unobserved state space, instead of quadratic. However, it is unlikely that further restrictions could reduce the complexity to be sublinear in the size of the joint unobserved state space without overly restricting the representation.

7.1.3 PSDG Domain Specification

The PSDG formalism cannot perform the reflexive modeling required when the recognizing agent's decision affects the observed agent's planning process. For example, in the clearly adversarial domain of air combat, existing agent tracking research uses a recursive approach to explicitly model the observed agent's beliefs about the recognizing agent. The PSDG inference algorithms treat the domain specification as fixed throughout the entire recognition process. In domains where the observed agent may modify its behavior, the recognizing agent could profit from dynamically altering its PSDG model. There currently exist PCFG learning algorithms [24] that can generate the productions and production probabilities based on labeled parse trees. Analogous algorithms could generate PSDG productions, states, and probabilities based on labeled parse trees. In a dynamic plan recognition session, it is unclear what source would provide the labels, but such algorithms could update state transition probabilities at the very least. A PSDG learning algorithm would also simplify the initial domain specification process. In domains where the observed agent's behavior depends significantly on the recognizing agent's decisions, even a dynamic PSDG specification may be too weak to model this reflexive dependency. However, such dependencies cause difficulties for existing plan recognition approaches, although the work on learning domain specifications offers a potential solution [30, 2].

7.2 Contributions

The general plan recognition framework presented in Chapter 2 provides the basis for the contributions of this dissertation. The goal of this research has been to develop a probabilistic language for modeling problem domains, following the outline of Figure 2.1. This general outline represents the key components of all plan recognition problems, but an operational mechanism requires a restricted language. In the search for such a language, this line of research began with the generality of the Bayesian network representation.

Hand-coding Bayesian networks is impractical as a general approach to plan recognition, so we considered the more restrictive language of the PCFG model. The algorithms presented in Chapter 3 automatically generate a Bayesian network representing the distribution over all parses of strings (bounded in length by some parameter) in the language of a PCFG. Using the standard Bayesian network inference algorithms, we can compute the conditional probability or most probable configuration of any collection of our basic random

variables, given any other event which can be expressed in terms of these variables. Although answering queries in Bayesian networks is exponential in the worst case, our method incurs this cost in the service of greatly increased generality. Our hope is that the enhanced scope will make PCFGs a useful model for domains that require more flexibility in query forms and in probabilistic structure, while also extending the usefulness of PCFGs in natural language processing and other pattern recognition domains where they have already been successful.

However, the PCFG model, even with the extensions considered in Chapter 4, is unsuitable for most plan recognition domains. The analysis of the PCFG model's limitations provided the motivation behind the central contribution of this dissertation, the PSDG model presented in Chapter 5. This new representation language supports many of the features desirable when modeling plan generation. The underlying plan grammar is capable of representing the plan/action hierarchy through the production structure, although the decomposition production format requires a total order over subplans. The state variables can capture the agent's mental state, as well as the external context from which that mental state derives. The state variables allow us to model the dependency of plan selection on the agent's beliefs, preferences, and capabilities through conditional production probabilities. The state variables also represent the world dynamics, allowing us to model the dependency of future plan contexts on the agent's current action, though not on higher-level subplans.

The PSDG model supports the generation of a Bayesian network representation of an underlying distribution, and we can use such networks to compute conditional probabilities over any combination of plan/state variables. However, the generated networks are likely to be intractable for all but the simplest PSDGs. The PSDG-specific inference algorithms scale much more efficiently than the Bayesian network algorithms, supporting practical inference in more significant planning domains, as demonstrated by the example domain specifications of Chapter 6. The PSDG-specific algorithms answer a more restricted set of queries, incorporating evidence about only the state variables, but still computing posterior distributions over both plan and state variables. The design choices inherent in the PSDG model and inference algorithms sacrifice the generality of some existing recognition approaches, but the efficiency gains derived from those assumptions will allow practical plan recognition in domains where those existing languages do not.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Jerome Azarewicz, Glenn Fala, Ralph Fink, and Christof Heithecker. Plan recognition for airborne tactical decision making. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 805–811, 1986.

[2] Mathias Bauer. Acquisition of abstract plan descriptions for plan recognition. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 936–941, Madison, WI, 1998. AAAI Press.

[3] Ezra Black, Fred Jelinek, John Lafferty, David M. Magerman, Robert Mercer, and Salim Roukos. Towards history-based grammars: Using richer models for probabilistic parsing. In Mitch Marcus, editor, *Proceedings of the Fifth DARPA Speech and Natural Language Workshop*, pages 31–37, Feb 1992.

[4] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 33–42, San Francisco, 1998. Morgan Kaufmann.

[5] Ted Briscoe and John Carroll. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59, 1993.

[6] Sandra Carberry. Incorporating default inferences into plan recognition. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 471–478, Boston, 1990. AAAI Press.

[7] Eugene Charniak. *Statistical Language Learning.* MIT Press, Cambridge, Mass., 1993.

[8] Eugene Charniak and Glenn Carroll. Context-sensitive statistics for improved grammatical language models. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 728–733, Menlo Park, Calif., 1994. AAAI Press.

[9] Eugene Charniak and Robert P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, November 1993.

[10] Eugene Charniak and Solomon Eyal Shimony. Cost-based abduction and MAP explanation. *Artificial Intelligence*, 66:345–374, 1994.

[11] Philip A. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. In *Proceedings of SPIE: Visual Communications and Image Processing IV*, pages 852–863, Bellingham, Wash., 1989. International Society for Optical Engineering.

[12] Adnan Darwiche and Gregory Provan. Query DAGs: A practical paradigm for implementing belief-network inference. *Journal of Artificial Intelligence Research*, 6:147–176, 1997.

[13] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 211–219, San Francisco, 1996. Morgan Kaufmann.

[14] Rina Dechter. Topological parameters for time-space tradeoff. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 220–227, San Francisco, 1996. Morgan Kaufmann.

[15] Jeff Forbes, Tim Huang, Keiji Kanazawa, and Stuart Russell. The BATmobile: Towards a Bayesian automated taxi. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1878–1885, 1995.

[16] Robert P. Goldman, Christopher W. Geib, and Christopher A. Miller. An extended model for probabilistic plan recognition. Draft.

[17] Rafael C. Gonzalez and Michael S. Thomason. *Syntactic Pattern Recognition: An Introduction.* Addison-Wesley, Reading, Mass., 1978.

[18] Barbara J. Grosz and Candace L. Sidner. Plans for discourse. In Philip R. Cohen, Jerry Morgan, and Martha E. Pollack, editors, *Intentions in Communication*, pages 417–444. MIT Press, Cambridge, MA, 1990.

[19] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass., 1979.

[20] Marcus J. Huber and Edmund H. Durfee. Observational uncertainty in plan recognition among interacting robots. In *Working Notes: Workshop on Dynamically Interacting Robots*, pages 68–75, Chambery, France, 1993.

[21] Marcus J. Huber, Edmund H. Durfee, and Michael P. Wellman. The automated mapping of plans for plan recognition. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 344–351, 1994.

[22] Francois F. Ingrand, Michael P. Georgeff, and Anand S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6):34–44, December 1992.

[23] Anthony Jameson. Numerical uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User-Adapted Interaction*, 5(3-4):193–251, 1995.

[24] Frederick Jelinek, John D. Lafferty, and R. L. Mercer. Basic methods of probabilistic context free grammars. In P. Laface and R. DeMori, editors, *Speech Recognition and Understanding*, pages 345–360. Springer, Berlin, 1992.

[25] Finn V. Jensen. *An Introduction to Bayesian Networks.* Springer, New York, 1996.

[26] Henry A. Kautz and James F. Allen. Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 32–37, 1986.

[27] Uffe Kjærulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence*, pages 121–129, San Mateo, CA, 1992. Morgan Kaufmann.

[28] Daphne Koller, David McAllester, and Avi Pfeffer. Effective Bayesian inference for stochastic programs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 740–747, Menlo Park, Calif., 1997. AAAI Press.

[29] Jaeho Lee, Marcus J. Huber, Edmund H. Durfee, and Patrick G. Kenny. UM-PRS: An implementation of the procedural reasoning system for multirobot applications. In *Proceedings of the AIAA/NASA Conference on Intelligent Robotics in Field, Factory, Service, and Space*, pages 842–849, March 1994.

[30] Neal Lesh and Oren Etzioni. Scaling up goal recognition. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, pages 178–189, 1996.

[31] Dekang Lin and Randy Goebel. A message passing algorithm for plan recognition. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 280–285, 1991.

[32] David M. Magerman and Mitchell P. Marcus. Pearl: A probabilistic chart parser. In *Proceedings of the Second International Workshop on Parsing Technologies*, pages 193–199, 1991.

[33] Richard E. Neapolitan. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. John Wiley and Sons, New York, 1990.

[34] Hermann Ney. Stochastic grammars and pattern recognition. In P. Laface and R. DeMori, editors, *Speech Recognition and Understanding*, pages 319–344. Springer, Berlin, 1992.

[35] Stephen J. Payne and T. R. G. Green. Task-action grammars: A model of the mental representation of task languages. *Human-Computer Interaction*, 2:93–133, 1986.

[36] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1987.

[37] David V. Pynadath and Michael P. Wellman. Accounting for context in plan recognition, with application to traffic monitoring. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 472–481, San Francisco, 1995. Morgan Kaufmann.

[38] David V. Pynadath and Michael P. Wellman. Generalized queries on probabilistic context-free grammars. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):65–77, 1998.

[39] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[40] Yasubumi Sakakibara, Michael Brown, Rebecca C. Underwood, I. Saira Mian, and David Haussler. Stochastic context-free grammars for modeling RNA. In *Proceedings*

of the 27th Hawaii International Conference on System Sciences, pages 284–293, Los Alamitos, Calif., 1995. IEEE Computer Society Press.

[41] Milind Tambe. Recursive agent and agent-group tracking in a real-time, dynamic environment. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 368–375, 1995.

[42] Milind Tambe and Paul S. Rosenbloom. Event tracking in a dynamic multi-agent environment. *Computational Intelligence*, 12(3):499–521, 1996.

[43] Marc Vilain. Getting serious about parsing plans: A grammatical analysis of plan recognition. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 190–197, Boston, 1990. AAAI Press.

[44] Michael P. Wellman. The STRIPS assumption for planning under uncertainty. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 198–203, Boston, 1990. AAAI Press.

[45] C. S. Wetherell. Probabilistic languages: A review and some open questions. *Computing Surveys*, 12(4):361–379, 1980.

APPENDICES

APPENDIX A

PSDG Representation of Traffic Domain

A.1 Nonterminal Symbols

Drive start symbol, the driver's top-level plan

Pass passing maneuver

StartLeft initiates left lane change when conditions allow

StartRight initiates right lane change when conditions allow

Left moves car one lane to the left

Right moves car one lane to the right

StartLeftAcc initiates left lane change with acceleration when conditions allow

StartRightAcc initiates right lane change with acceleration when conditions allow

LeftAcc moves car one lane to the left and increases speed

RightAcc moves car one lane to the right and increases speed

2Left moves car two lanes to the left

2Right moves car two lanes to the right

Stay keeps car in current lane

ChooseAcc high-level plan for determining change of speed

A.2 Terminal Symbols

MoveL moves car to the left

MoveR moves car to the right

NoMove car does not move laterally

Accelerate car increases speed

Decelerate car decreases speed

NoAccelerate car maintains current speed

MoveL moves car to the left

MoveR moves car to the right

NoMove car does not move laterally

SignalL signals to the left

SignalR signals to the right

SignalOff no signal

Exit car leaves highway

A.3 State Variables

Xpos lateral position of observed car, in $\{0, 1, \ldots, 19\}$ units of feet, right lane corresponds to Xpos of 0–6, middle lane 7–12, left lane 13–19. Observed.

Ypos position of observed car along highway, in $\{0.00, 0.01, 0.02, \ldots, 19.98, 19.99, 20.00\}$ units of miles. Observed.

Speed speed of observed car (not counting lateral movement), in $\{40, 45, \ldots, 90, 95\}$ units of mph. Observed.

Signal state of observed car's turn indicator, in $\{\mathsf{off}, \mathsf{left}, \mathsf{right}\}$. Observed.

FL-car speed of car in left lane to the front of observed car, with same domain as Speed. Observed.

FM-car speed of car in middle lane to the front of observed car, with same domain as Speed. Observed.

FR-car speed of car in right lane to the front of observed car, with same domain as Speed. Observed.

ML-car speed of car in left lane and even with observed car, with same domain as Speed. Observed.

MM-car speed of car in middle lane and even with observed car, with same domain as Speed. Observed.

MR-car speed of car in right lane and even with observed car, with same domain as Speed. Observed.

BL-car speed of car in left lane and behind observed car, with same domain as Speed. Observed.

BM-car speed of car in middle lane and behind observed car, with same domain as Speed. Observed.

BR-car speed of car in right lane and behind observed car, with same domain as Speed. Observed.

Exit intended exit of observed car, in $\{4.00, 8.00, \ldots, 20.00\}$. Unobserved.

DesiredSpeed preferred driving speed of observed car, in $\{50, 55, \ldots, 95\}$. Unobserved.

Type cautiousness of observed driver, in {normal, cautious, aggressive}. Unobserved.

A.4 Productions

The production probability functions for the expansions of a given nonterminal symbol use a similarly structured decision procedure, so we list only a single probability function covering the productions for a nonterminal symbol, but with an additional argument prod to specify which production is currently under consideration. The accelerating versions of the lane change symbols use the same production probability functions as the symbols with unspecified accelerations, although with corresponding changes to the prod argument.

1. Drive $\rightarrow$ Stay Drive (p_d)

2. Drive → StartLeft Drive (p_d)

3. Drive → StartRight Drive (p_d)

4. Drive → 2Left Drive (p_d)

5. Drive → 2Right Drive (p_d)

6. Drive → Pass Drive (p_d)

7. Drive → Exit (p_d)

```
    if (Xpos < 7)       /* Right lane */
      lane = 2;
    else if (Xpos < 13) /* Middle lane */
      lane = 1;
    else              /* Left lane */
      lane = 0;
    if (Ypos >= exit) /* If at exit, then exit right now */
      if (prod == 7) /* Exit */
        return(1.0);
      else
        return(0.0);
    else if (prod == 6) /* Exit under no other circumstance */
      return(0.0);
    else if (Ypos > Exit-0.5) /* Within exiting area of desired exit */
      switch (prod) {
      case 0: /* Stay */
        if (lane == 2)
  return(1.0);
        else
  return(0.0);
      case 2: /* Right */
        if (lane == 1)
  return(1.0);
        else
  return(0.0);
```

```
    case 4: /* 2-Right */
      if (lane == 0)
return(1.0);
      else
return(0.0);
    default:
      return(0.0);
    }
  else if ((F(lane)-car is null) ||
   (F(lane)-car >= DesiredSpeed)) /* Front is unobstructed */
    switch (prod) {
    case 0: /* Stay */
      return(0.9);
    case 1: /* Left */
      if (lane != 0)
return(0.05);
      else
return(0.0);
    case 2: /* Right */
      if (lane != 2)
return(0.05);
      else
return(0.0);
    case 3: /* 2-Left */
      if (lane == 2)
return(0.05);
      else
return(0.0);
    case 4: /* 2-Right */
      if (lane == 0)
return(0.05);
      else
return(0.0);
```

```
    default:
      return(0.0);
    }
  else {
   /* Compute average traveling speed in each lane */

    for (X=L,M,R) {
      k = speeds[X] = 0;
      for (Y=B,M,F)
if (YX-car is not nul) {
  k++;
  speeds[X] += cars[Y][X];
}
      if (k == 0)
speeds[X] = DesiredSpeed;
      else
speeds[X] /= k;
    }
    /* Choose target lane with most desirable traveling speed */
    TargetLane = lane;
    for (X=R,M,L)  /* Favoring lanes toward the right */
      if (abs(speeds[X]-DesiredSpeed) <
          abs(speeds[TargetLane]-DesiredSpeed))
TargetLane = X;
    switch (prod) {
    case 0: /* Stay */
      return(0.0);
    case 1: /* Left */
      if (TargetLane == lane - 1)
return(1.0);
      else
return(0.0);
    case 2: /* Right */
```

```
      if (TargetLane == lane + 1)
return(1.0);
      else
return(0.0);
    case 3: /* 2-Left */
      if (TargetLane == lane - 2)
return(1.0);
      else
return(0.0);
    case 4: /* 2-Right */
      if (TargetLane == lane + 2)
return(1.0);
      else
return(0.0);
    case 5: /* Pass */
      if (TargetLane == lane)
return(1.0);
      else
return(0.0);
    }
  }
```

8. Pass $\rightarrow$ Stay (p_p)
9. Pass $\rightarrow$ Stay Pass (p_p)
10. Pass $\rightarrow$ LeftAcc RightAcc (p_p)
11. Pass $\rightarrow$ RightAcc LeftAcc (p_p)

```
  if (Xpos < 7)       /* Right lane */
    lane = 2;
  else if (Xpos < 13) /* Middle lane */
    lane = 1;
  else             /* Left lane */
    lane = 0;
```

```
  if (Ypos > Exit-0.5)
    return((prod==8)?1.0:0.0);
  else if ((F(lane)-car is null) ||
   (F(lane)-car >= DesiredSpeed)) /* Front is unobstructed */
    return((prod==8)?1.0:0.0);
  else if (prod == 8)
    return(0.0);
  else if ((lane == 0) && (prod == 10))
    return(0.0);
  else if ((lane == 2) && (prod == 11))
    return(0.0);
  else {
    LeftFree = (((B(left of lane)-car is null) ||
  ((Type != cautious) && (B(left of lane)-car < Speed))) &&
 (M(left of lane)-car is not null) &&
  (F(left of lane)-car is null) ||
  ((Type == normal) && (B(left of lane)-car > Speed)) ||
  (Type == aggressive)));
    RightFree = (((B(right of lane)-car is null) ||
  ((Type != cautious) && (B(right of lane)-car < Speed))) &&
 (M(right of lane)-car is not null) &&
   ((F(right of lane)-car is null) ||
  ((Type == normal) && (B(right of lane)-car > Speed)) ||
  (Type == aggressive)));
    if (LeftFree) {
      if (RightFree)
switch (prod) {
case 9: /* Stay */
  return(0.01);
case 10: /* Pass on left */
  switch(Type) {
  case normal:
    return(0.95);
```

```
  case cautious:
    return(0.98);
  case aggressive:
    return(0.9);
  }
case 11: /* Pass on right */
  switch(Type) {
  case normal:
    return(0.04);
  case cautious:
    return(0.01);
  case aggressive:
    return(0.09);
  }
}
      else
switch (prod) {
case 9: /* Stay */
  return(0.005);
case 10: /* Pass on left */
  switch(Type) {
  case normal:
    return(0.993);
  case cautious:
    return(0.994);
  case aggressive:
    return(0.99);
  }
case 11: /* Pass on right */
  switch(Type) {
  case normal:
    return(0.002);
  case cautious:
```

```
    return(0.001);
  case aggressive:
    return(0.005);
  }
}
    }
    else if (RightFree)
      switch (prod) {
      case 9: /* Stay */
switch(Type) {
case normal:
  return(0.2);
case cautious:
  return(0.5);
case aggressive:
  return(0.1);
}
      case 10: /* Pass on left */
  return(0.01);
      case 11: /* Pass on right */
switch(Type) {
case normal:
  return(0.79);
case cautious:
  return(0.49);
case aggressive:
  return(0.89);
}
      }
    else
      switch (prod) {
      case 9: /* Stay */
switch(Type) {
```

```
case normal:
  return(0.99);
case cautious:
  return(0.999);
case aggressive:
  return(0.9);
}
return(0.005);
      case 10: /* Pass on left */
switch(Type) {
case normal:
  return(0.009);
case cautious:
  return(0.0009);
case aggressive:
  return(0.03);
}
      case 11: /* Pass on right */
switch(Type) {
case normal:
  return(0.001);
case cautious:
  return(0.0001);
case aggressive:
  return(0.07);
}
      }
  }
```

12. StartLeft → Stay (p_{sl})
13. StartLeft → Stay StartLeft (p_{sl})
14. StartLeft → MoveL ChooseAcc SignalL Left (p_{sl})
15. StartLeft → MoveL ChooseAcc SignalOff Left (p_{sl})

16. StartLeft → MoveL ChooseAcc SignalR Left (p_{sl})

```
  if ( < 7)       /* Right lane */
    lane = 2;
  else if (Xpos < 13) /* Middle lane */
    lane = 1;
  else             /* Left lane */
    lane = 0;
  if (Ypos > Exit-0.5)
    return((prod==12)?1.0:0.0);
  else if (prod == 12)
    return(0.0);
  else
    switch (Type) {
    case normal:
      if ((M(left of lane)-car is null) &&
          ((B(left of lane)-car is null) ||
           (B(left of lane)-car < Speed)))
switch (prod) {
case 13: return(0.1);
case 14: return(0.85);
case 15: return(0.049);
case 16: return(0.001);
}
      else
switch (prod) {
case 13: return(0.9);
case 14: return(0.095);
case 15: return(0.0049);
case 16: return(0.0001);
}
    case cautious:
      if ((M(left of lane)-car is null) &&
          (B(left of lane)-car is null))
```

```
switch (prod) {
case 13: return(0.1);
case 14: return(0.85);
case 15: return(0.049);
case 16: return(0.001);
}
      else
switch (prod) {
case 13: return(0.9);
case 14: return(0.095);
case 15: return(0.0049);
case 16: return(0.0001);
}
    case aggressive:
      if ((M(left of lane)-car is null) &&
          ((B(left of lane)-car is null) ||
           (B(left of lane)-car < Speed)))
switch (prod) {
case 13: return(0.1);
case 14: return(0.85);
case 15: return(0.049);
case 16: return(0.001);
}
      else
switch (prod) {
case 13: return(0.9);
case 14: return(0.095);
case 15: return(0.0049);
case 16: return(0.0001);
}
    }
}
```

17. StartRight $\rightarrow$ Stay (p_{sr})

18. StartRight → Stay StartRight (p_{sr})

19. StartRight → MoveR ChooseAcc SignalL Right (p_{sr})

20. StartRight → MoveR ChooseAcc SignalOff Right (p_{sr})

21. StartRight → MoveR ChooseAcc SignalR Right (p_{sr})

```
  if ( < 7)       /* Right lane */
    lane = 2;
  else if (Xpos < 13) /* Middle lane */
    lane = 1;
  else             /* Left lane */
    lane = 0;
  if (Ypos > Exit-0.5)
    return((prod==17)?1.0:0.0);
  else if (prod == 17)
    return(0.0);
  else
    switch (Type) {
    case normal:
      if ((M(right of lane)-car is null) &&
          ((B(right of lane)-car is null) ||
           (B(right of lane)-car < Speed)))
switch (prod) {
case 18: return(0.1);
case 19: return(0.85);
case 20: return(0.049);
case 21: return(0.001);
}
      else
switch (prod) {
case 18: return(0.9);
case 19: return(0.095);
case 20: return(0.0049);
case 21: return(0.0001);
```

```
}
    case cautious:
      if ((M(right of lane)-car is null) &&
          (B(right of lane)-car is null))
switch (prod) {
case 18: return(0.1);
case 19: return(0.85);
case 20: return(0.049);
case 21: return(0.001);
}
      else
switch (prod) {
case 18: return(0.9);
case 19: return(0.095);
case 20: return(0.0049);
case 21: return(0.0001);
}
    case aggressive:
      if ((M(right of lane)-car is null) &&
          ((B(right of lane)-car is null) ||
           (B(right of lane)-car < Speed)))
switch (prod) {
case 18: return(0.1);
case 19: return(0.85);
case 20: return(0.049);
case 21: return(0.001);
}
      else
switch (prod) {
case 18: return(0.9);
case 19: return(0.095);
case 20: return(0.0049);
case 21: return(0.0001);
```

```
}
    }
}
```

22. Left → Stay (p_l)

23. Left → MoveL ChooseAcc SignalL Left (p_l)

24. Left → MoveL ChooseAcc SignalOff Left (p_l)

25. Left → MoveL ChooseAcc SignalR Left (p_l)

```
  if (Xpos < 7)       /* Right lane */
    lane = 2;
  else if (Xpos < 13) /* Middle lane */
    lane = 1;
  else             /* Left lane */
    lane = 0;
  if ((Xpos == 16) || (Xpos == 10) || (Xpos == 3))
    if (prod == 22)
return(1.0);
    else
      return(0.0);
  else if (Signal == left)
    switch (prod) {
    case 22: return(0.0);
    case 23: return(0.999);
    case 24: return(0.0009);
    case 25: return(0.0001);
    }
  else if (Signal == off)
    switch (prod) {
    case 22: return(0.0);
    case 23: return(0.0099);
    case 24: return(0.99);
    case 25: return(0.0001);
```

```
    }
  else
    switch (prod) {
    case 22: return(0.0);
    case 23: return(0.01);
    case 24: return(0.89);
    case 25: return(0.1);
    }
```

26. Right $\rightarrow$ Stay (p_r)

27. Right $\rightarrow$ MoveR ChooseAcc SignalL Right (p_r)

28. Right $\rightarrow$ MoveR ChooseAcc SignalOff Right (p_r)

29. Right $\rightarrow$ MoveR ChooseAcc SignalR Right (p_r)

```
  if (Xpos < 7)       /* Right lane */
    lane = 2;
  else if (Xpos < 13) /* Middle lane */
    lane = 1;
  else             /* Left lane */
    lane = 0;
  if ((Xpos == 16) || (Xpos == 10) || (Xpos == 3))
    if (prod == 22)
return(1.0);
    else
      return(0.0);
  else if (Signal == right)
    switch (prod) {
    case 22: return(0.0);
    case 23: return(0.999);
    case 24: return(0.0009);
    case 25: return(0.0001);
    }
  else if (Signal == off)
```

```
    switch (prod) {
    case 22: return(0.0);
    case 23: return(0.0099);
    case 24: return(0.99);
    case 25: return(0.0001);
    }
  else
    switch (prod) {
    case 22: return(0.0);
    case 23: return(0.01);
    case 24: return(0.89);
    case 25: return(0.1);
    }
```

30. StartLeftAcc → Stay (p_{sl})
31. StartLeftAcc → Stay StartLeftAcc (p_{sl})
32. StartLeftAcc → MoveL Accelerate SignalL LeftAcc (p_{sl})
33. StartLeftAcc → MoveL Accelerate SignalOff LeftAcc (p_{sl})
34. StartLeftAcc → MoveL Accelerate SignalR LeftAcc (p_{sl})
35. StartRightAcc → Stay (p_{sr})
36. StartRightAcc → Stay StartRightAcc (p_{sr})
37. StartRightAcc → MoveR Accelerate SignalL RightAcc (p_{sr})
38. StartRightAcc → MoveR Accelerate SignalOff RightAcc (p_{sr})
39. StartRightAcc → MoveR Accelerate SignalR RightAcc (p_{sr})
40. LeftAcc → Stay (p_l)
41. LeftAcc → MoveL Accelerate SignalL LeftAcc (p_l)
42. LeftAcc → MoveL Accelerate SignalOff LeftAcc (p_l)
43. LeftAcc → MoveL Accelerate SignalR LeftAcc (p_l)

44. RightAcc → Stay (p_r)

45. RightAcc → MoveR Accelerate SignalL RightAcc (p_r)

46. RightAcc → MoveR Accelerate SignalOff RightAcc (p_r)

47. RightAcc → MoveR Accelerate SignalR RightAcc (p_r)

48. 2Left → StartLeft StartLeft (1.0)

49. 2Right → StartRight StartRight (1.0)

50. Stay → NoMove ChooseAcc SignalL (p_s)

51. Stay → NoMove ChooseAcc SignalOff (p_s)

52. Stay → NoMove ChooseAcc SignalR (p_s)

53. ChooseAcc → Accelerate (p_a)

54. ChooseAcc → Decelerate (p_a)

55. ChooseAcc → NoAccelerate (p_a)

```
if (Xpos < 7)       /* Right lane */
  lane = 2;
else if (Xpos < 13) /* Middle lane */
  lane = 1;
else             /* Left lane */
  lane = 0;
if (F(lane)-car < Speed)
  return((prod==51)?1.0:0.0);
else if (Speed < DesiredSpeed)
  return((prod==49)?1.0:0.0);
else if (Speed > DesiredSpeed)
  if (B(lane)-car > Speed)
    return((prod==50)?1.0:0.0);
  else
    return((prod==51)?1.0:0.0);
else
  return((prod==50)?1.0:0.0);
```

A.5 Prior Probability Distribution

Xpos The driver starts in the middle of one of the three lanes:

x	$\Pr(\mathsf{Xpos}(Q^0) = x)$
3	1/3
10	1/3
16	1/3
otherwise	0

Ypos The driver starts at the beginning of the highway, so $\Pr(\mathsf{Ypos}(Q^0) = 0) = 1$.

Speed The driver's speed is uniformly distributed, so $\Pr(\mathsf{Speed}(Q^0) = v) = 1/12$

XX-Car The values for the other cars are uniformly distributed.

Exit The driver's intended exit is uniformly distributed, so $\Pr(\mathsf{Exit}(Q^0) = y) = 1/5$.

DesiredSpeed The driver's preferred traveling speed is distributed as follows:

v	$\Pr(\mathsf{DesiredSpeed}(Q^0) = v)$
50	0.02
55	0.03
60	0.05
65	0.15
70	0.23
75	0.27
80	0.15
85	0.05
90	0.03
95	0.02

Type The driver's type is distributed as follows:

t	$\Pr(\mathsf{Type}(Q^0) = t)$
normal	0.75
cautious	0.10K
aggressive	0.15

A.6 World Dynamics

Xpos If the driver performs a MoveL, then its lateral position evolves as follows:

x_1	$\Pr(\mathsf{Xpos}(Q^1) = x_1 \mid \mathsf{Xpos}(Q^0) = x_0, \Sigma^1 = \mathsf{MoveL})$
$x_0 + 3$	0.1
$x_0 + 2$	0.3
$x_0 + 1$	0.6

If the driver performs a MoveR, then its lateral position evolves as follows:

x_1	$\Pr(\mathsf{Xpos}(Q^1) = x_1 \mid \mathsf{Xpos}(Q^0) = x_0, \Sigma^1 = \mathsf{MoveR})$
$x_0 - 3$	0.1
$x_0 - 2$	0.3
$x_0 - 1$	0.6

If the driver performs neither a MoveL nor a MoveR, then its lateral position does not change, so $\Pr(\mathsf{Xpos}(Q^1) = x_0 \mid \mathsf{Xpos}(Q^0) = x_0, \Sigma^1 \notin \{\mathsf{MoveL}, \mathsf{MoveR}\}) = 1$.

Yprob The driver's new position along the highway is independent of low-level actions, but we condition on the acceleration maneuver, so that we update the position only once for the three components. In the following expression, we define AccManeuver$\equiv$ {Accelerate, Decelerate, NoAccelerate}:

$$\Pr(\mathsf{Ypos}(Q^1) \approx y_0 + 2 * v_0/3600 \mid \mathsf{Ypos}(Q^0) = y_0, \mathsf{Speed}(Q^0) = v_0, \Sigma^1 \in \mathsf{AccManeuver}) = 1$$

For the other two components,

$$\Pr(\mathsf{Ypos}(Q^1) = y_0 \mid \mathsf{Ypos}(Q^0) = y_0, \Sigma^1 \notin \mathsf{AccManeuver}) = 1$$

Speed The driver's speed changes as follows:

Σ^1	v_1	$\Pr(\mathsf{Speed}(Q^1) = v_1 \mid \mathsf{Speed}(Q^0) = v_0, \Sigma^1)$
Accelerate	$v_0 + 15$	0.1
Accelerate	$v_0 + 10 = 95$	0.33
Accelerate	$v_0 + 10 < 95$	0.3
Accelerate	$v_0 + 5 = 95$	1.0
Accelerate	$v_0 + 5 = 90$	0.67
Accelerate	$v_0 + 5 < 90$	0.6
Accelerate	$v_0 = 95$	1.0
Accelerate	$v_0 \neq 95$	0.0
Accelerate	$< v_0$	0.0
Decelerate	$v_0 - 15$	0.1
Decelerate	$v_0 - 10 = 40$	0.33
Decelerate	$v_0 - 10 > 40$	0.3
Decelerate	$v_0 - 5 = 40$	1.0
Decelerate	$v_0 - 5 = 45$	0.67
Decelerate	$v_0 - 5 > 45$	0.6
Decelerate	$v_0 = 40$	1.0
Decelerate	$v_0 \neq 40$	0.0
Decelerate	$> v_0$	0.0
NoAccelerate	v_0	1.0

XX-Car The variables for the other cars have a uniform distribution, independent of any other variables.

Exit,DesiredSpeed,Type These preferences are time-invariant.

APPENDIX B

PSDG Representation of Air Combat

B.1 Nonterminal Symbols

ExecuteMission start symbol, pilot's top-level plan

FlyToTarget pilot simply flies to chosen target

Intercept pilot engages enemy plane

BugOut

Wait delay plan, used while waiting for certain conditions to hold

EmployWeapons pilot fires missile at enemy plane

Evade pilot evades actions by enemy plane

Fpole turning maneuver useful in guiding missile

SelMissile select missile to fire, simply a delay in this model

GetMissileLAR plan reach launch acceptability region

FpoleLeft guidance maneuver with direction specified

FpoleRight guidance maneuver with direction specified

StartFpoleLeft initiates guidance maneuver

StartFpoleRight initiates guidance maneuver

LaunchMissile plan for launching missile at enemy plane

GetSteeringCircle plan to achieve steering circle in preparation for firing missile

LockMissile plan to lock enemy target

BeamLeft evasive maneuver to left

BeamRight evasive maneuver to right

StartBeamLeft initiates evasive maneuver

StartBeamRight initiates evasive maneuver

B.2 Terminal Symbols

LeftTurn turns plane to the left

RightTurn turns plane to the right

MaintainHeading keeps plane at current heading

FireMissile fires missile at enemy plane

B.3 State Variables

EnemyMissilep Boolean indicating whether the enemy plane has a fired a missile at observed plane. Observed.

Bogeyp Boolean indicating whether there is an enemy plane or not. Observed.

MyHeading Observed pilot's heading, in $\{0, 15, 30, \ldots, 345\}$, in degrees. Observed.

EnemyXPos X coordinate of enemy's position, relative to observed pilot, in {-10,-9,. . . ,0, . . . , 10}. Observed

EnemyYPos Y coordinate of enemy's position, relative to observed pilot, in {-10,-9,. . . ,0, . . . , 10}. Observed.

EnemyHeading Enemy pilot's heading, in $\{0, 15, 30, \ldots, 345\}$, in degrees. Observed.

EnemyTurning indicates whether enemy pilot is currently changing its heading, in {none, left, right}. Observed.

Additional variables InLARp (Boolean), EnemyTheta (in degrees), and EnemyBearing ({straight, left, right}), are useful in simplifying the descriptions of productions and world dynamics. All three are completely deterministic given the values of the state variables listed.

B.4 Productions

The production probability functions specify only nonzero values.

1. ExecuteMission $\rightarrow$ FlyToTarget ExecuteMission ($p_1(q) = 1.0$ if Bogeyp(q) = false
2. ExecuteMission $\rightarrow$ BugOut ExecuteMission ($p_2(q) = 0.1$ if Bogeyp(q) = true)
3. ExecuteMission $\rightarrow$ Intercept ExecuteMission ($p_3(q) = 0.9$ if Bogeyp(q) = true)
4. FlyToTarget $\rightarrow$ MaintainHeading ($p_4(q) = 1.0$)
5. BugOut $\rightarrow$ LeftTurn BugOut ($p_5(q) = 1.0$ if Bogeyp = true, EnemyBearing = right)
6. BugOut $\rightarrow$ RightTurn BugOut ($p_6(q) = 1.0$ if Bogeyp = true, EnemyBearing = left)
7. BugOut $\rightarrow$ MaintainHeading BugOut ($p_7(q) = 1.0$ if Bogeyp = true, EnemyBearing = straight)
8. BugOut $\rightarrow$ MaintainHeading ($p_8(q) = 1.0$ if Bogeyp = false)
9. Intercept $\rightarrow$ Evade Intercept ($p_9(q) = 1.0$ if EnemyMissilep(q) = true, Bogeyp = true)
10. Intercept $\rightarrow$ EmployWeapons Intercept ($p_{10}(q) = 1.0$ if EnemyMissilep(q)=false, Bogeyp=true)
11. Intercept $\rightarrow$ MaintainHeading ($p_{11}(q) = 1.0$ if Bogeyp = false)
12. EmployWeapons $\rightarrow$ SelMissile GetMissileLAR Launch-missile Fpole ($p_{12}(q) = 1.0$)
13. SelMissile $\rightarrow$ MaintainHeading ($p_{13}(q) = 1.0$)
14. GetMissileLAR $\rightarrow$ RightTurn GetMissileLAR ($p_{14}(q) = 1.0$ if InLARp=false, EnemyBearing=right, EnemyMissilep(q)=false)
15. GetMissileLAR $\rightarrow$ LeftTurn GetMissileLAR ($p_{15}(q) = 1.0$ if InLARp=false, EnemyBearing=left, EnemyMissilep(q)=false)

16. GetMissileLAR → MaintainHeading GetMissileLAR ($p_{16}(q) = 1.0$ if InLARp=false, EnemyBearing=straight, EnemyMissilep(q)= false)

17. GetMissileLAR → MaintainHeading ($p_{17}(q) = 1.0$ if InLARp=true, EnemyMissilep(q)= false)

18. GetMissileLAR → MaintainHeading ($p_{17}(q) = 1.0$ if EnemyMissilep(q) = true)

19. Launch-missile → GetSteeringCircle LockMissile FireMissile ($p_{18}(q) = 1.0$ if EnemyMissilep(q)=false)

20. Launch-missile → MaintainHeading ($p_{19}(q) = 1.0$ if EnemyMissilep(q)=true)

21. GetSteeringCircle → MaintainHeading ($p_{20}(q) = 1.0$ if EnemyBearing=straight, EnemyMissilep(q)=false)

22. GetSteeringCircle → RightTurn GetSteeringCircle ($p_{21}(q) = 1.0$ if EnemyBearing=right, EnemyMissilep(q)=false)

23. GetSteeringCircle → LeftTurn GetSteeringCircle ($p_{22}(q) = 1.0$ if EnemyBearing=left, EnemyMissilep(q)=false)

24. GetSteeringCircle → MaintainHeading ($p_{23}(q) = 1.0$ if EnemyMissilep(q)=true)

25. LockMissile → Wait Wait ($p_{24}(q) = 1.0$)

26. Wait → MaintainHeading ($p_{25}(q) = 1.0$)

27. Fpole → StartFpoleLeft FpoleLeft ($p_{26}(q) = 1.0$ if EnemyTurning = right)

28. Fpole → StartFpoleRight FpoleRight ($p_{27}(q) = 1.0$ if EnemyTurning = left)

29. Fpole → StartFpoleRight FpoleRight ($p_{28}(q) = 1.0$ if EnemyBearing = left)

30. Fpole → StartFpoleLeft FpoleLeft ($p_{29}(q) = 1.0$ if EnemyBearingright)

31. Fpole → StartFpoleRight FpoleRight ($p_{30}(q) = 0.5$ if EnemyBearingstraight)

32. Fpole → StartFpoleLeft FpoleLeft ($p_{31} = 0.5$ if EnemyBearingstraight)

33. StartFpoleLeft → LeftTurn ($p_{32}(q) = 1.0$)

34. StartFpoleRight → RightTurn ($p_{33}(q) = 1.0$)

35. FpoleRight → RightTurn RightTurn RightTurn ($p_{34}(q) = 1.0$)

36. FpoleLeft → LeftTurn LeftTurn LeftTurn ($p_{35}(q) = 1.0$)

37. Evade → StartBeamLeft BeamLeft ($p_{36}(q) = 1.0$ if EnemyBearing = right)

38. Evade → StartBeamRight BeamRight ($p_{37}(q) = 1.0$ if EnemyBearing = left)

39. Evade → StartBeamLeft BeamLeft ($p_{38}(q) = 0.5$ if EnemyBearing = straight)

40. Evade → StartBeamRight BeamRight ($p_{39}(q) = 0.5$ if EnemyBearing = straight)

B.5 Prior Probability Distribution

Bogeyp $\Pr(\text{Bogeyp}(Q^0)) = 0.6$

EnemyMissilep

B	$\Pr(\text{EnemyMissilep}(q) = \text{true} \mid Q^0 = q, \text{Bogeyp}(q) = B)$
true	0.3
false	0.0

MyHeading uniformly distributed

EnemyXPos if Bogeyp(q) =false, then the value is irrelevant, since there is no enemy plane. Otherwise,

x	$\Pr(\text{EnemyXPos}(Q^0) = x \mid Q^0, \text{Bogeyp}(Q^0))$
-10	9/70
-9	9/70
-8	7/70
-7	5/70
-6	3/70
-5	1/70
-4	1/70
-3	0
-2	0
-1	0
0	0
1	0
2	0
3	0
4	1/70
5	1/70
6	3/70
7	5/70
8	7/70
9	9/70
10	9/70

EnemyYPos same distribution as EnemyXPos

EnemyHeading if Bogeyp(Q^0) is true, then uniformly distributed; otherwise, irrelevant.

EnemyTurning

x	Pr(EnemyTurning(Q^0) $= x$)
straight	0.6
left	0.2
right	0.2

B.6 World Dynamics

Bogeyp We model the enemy planes with a fixed probability distribution.

Bogeyp(Q^0)	Pr(Bogeyp(Q^1)$\|Q^0$)
true	1.0
false	0.6

EnemyMissilep We model the enemy plane's missile firing with a fixed probability distribution.

EnemyMissilep(Q^0)	Bogeyp(Q^1)	Pr(EnemyMissilep(Q^1)$\|Q^0$, Bogeyp(Q^1))
·	false	0
true	true	1
false	true	0.3

MyHeading The low-level actions have a deterministic effect on the observed pilot's heading. If the pilot performs a RightTurn, its heading at time $t+1$ decreases by one step from the value at time t. Likewise, If the pilot performs a LeftTurn, its heading at time $t+1$ increases by one step from the value at time t. For all other actions, the heading remains unchanged.

EnemyTurning same as prior distribution

EnemyXPos same as prior distribution

EnemyYPos same as prior distribution

EnemyHeading same as prior distribution