Reasoning about Time and Probability

Keiji Kanazawa

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-92-61
May 1992

Reasoning about Time and Probability

by
Keiji Kanazawa
B. A., Bennington College, 1985
Sc. M., Brown University, 1988

Thesis
Submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy in the Department of Computer Science
at Brown University.

May 1992

This dissertation by Keiji Kanazawa
is accepted in its present form by the Department of
Computer Science as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Date _____ _____

Thomas L. Dean

Recommended to the Graduate Council

Date _____ _____

Eugene Charniak

Date _____ _____

Max Henrion (Stanford University)

Approved by the Graduate Council

Date _____ _____

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis is about how to picture the world and how to know what to do. It's about how to manage information about the world, and how to know the changes that might take place. It's about knowing what plan to commit to, and what action to undertake.

At the heart of this thesis is the idea that good decisions need good management of information about time, good ways of envisioning information about time. By envisioning, we mean the ability to hypothesize: to project into the future, or the past or present.

As an example, consider the selection of a travel route. Our task at hand might be the delivery of packages from Chicago to Atlanta. Or it may be a vacation drive through New England. In either case, the selection of travel route requires thinking and hypothesizing about many things; geographic distance, the time of day, the likelihood of delays and bottlenecks, and the scenery and amenities offered by each possible route.

An important part of envisioning is the ability to consider chance, the likelihood of different scenarios. For instance, we may know that Route 57 is more likely to result in delays during the afternoon. Thinking about chance includes the ability to extrapolate what the consequences of actions will be, given what we know about our current situation, and what we know about the likely future situation. For instance, even though Route 57 in general has a high chance of traffic jams, we may know that during certain times of the day, the likelihood is less; so if we leave after 3pm, say, we will miss traffic with a good chance.

Another important part of envisioning is the ability to judge the importance of the possible scenarios, their benefits and their liabilities. Through the process of

envisioning the importance and likelihoods of scenarios, we should know what the best action is. Traffic jams are unpleasant, but an alternate route may require traveling many more miles. Depending on our agenda, we may or may not decide to risk a one in ten chance of traffic jam.

In this thesis, we present a theory of how to envision such types of scenarios, and how to do so in a manner that is timely and useful. An informal statement of our problem is:

> *Let our knowledge about the world be a set of facts φ. The envisioning problem is the problem of deducing the set of facts ξ that will be true in the future, possibly as the result of our actions α. The decision problem is to select a good action $\hat{\alpha}$ from the set of actions in a manner that is timely and useful.*

We leave the question of what is good, timely, and useful vague for the moment. In our theory, we develop:

- A way to represent our knowledge about time, change, and chance.
- A way to efficiently extrapolate from our knowledge in order to decide what the best action is.

Our approach is distinguished by the following:

- We explicitly reason about chance with probability, and use stochastic process theory to reason about chance and time. Probability is a mature normative theory of reasoning about chance.
- We use logic to write down basic knowledge about probability and time. Logic makes it easy to quantify knowledge, and to apply knowledge as needed.
- We use graph models of knowledge about probability for the actual envisioning task. Graph models are a robust and efficient vehicle for envisioning with probability, and they make it easy to reason about chance from a normative basis.

The main results developed in this work are:

- We show how to perform a wide variety of reasoning about time and probability, including reasoning about lifetimes of facts, ordering of events, and time

dependent utility functions. Previous work in AI focused on state transition models that made such reasoning tasks difficult.

- We demonstrate the use of logic as the basis for a flexible language for building graph models about time. Thus we provide a computational theory for reasoning from the logic. Previous work either ignored time, or failed to bridge issues in knowledge representation with those of computation.

As part of this work, we have developed a series of programs for reasoning about time and probability. These include `Goo`, a *temporal database* for maintaining a picture of the likelihood of facts and events over time. `Goo` incrementally constructs and maintains graph models from logical knowledge in response to queries and assertions. Graph models are used to answer queries such as "What is the chance that I will arrive by noon?", and "Is there any plan where the probability of traffic jams is less than .5?".

The chief contribution of this work is in effecting a practical and coherent synthesis of ideas bridging representation in logic and inference in graph models. The underlying structure of knowledge and theories about time, change, and uncertainty are shared by the logic and graph models, making it easy to move from one to another. This transparent connection promises a wider applicability of methods of temporal reasoning under uncertainty in practical problem solving domains. We envision the application of such *temporal information management* technology in a wide variety of domains. Some applications are large scale – managing information about air traffic or cargo transport. Some are smaller – controlling some facet of the behavior of a 10kg mobile robot.

In addition to the chief contribution of presenting a coherent approach to reasoning about time and chance encompassing theory and experimentation, this dissertation makes specific contributions in the design of:

- Logics.
- Graph models.
- Applications in planning and control.
- Efficient implementations.

In this chapter, we provide a summary of our theory, as well as the organization of the thesis. Since our aim is to provide an overview of the issues involved, we will defer citations for the most part to the actual chapters in which the issues are discussed.

Time and Change

A major component of our theory is a rethinking of commonsense reasoning about time, change, and causation by way of probability. There are good reasons for the use of probability in reasoning about time and change:

- Probability is a language that directly addresses the notions of chance and likelihood.
- Probability is a *normative* theory of reasoning about chance.
- Probability is a mature theory, with clean and transparent semantics.
- By using probability, we open the way to use of *decision theory* for selecting actions.

Cox [1946] has shown that any calculus about chance and likelihood that adheres to a minimal set of desiderata ends up looking exactly like probability theory. Thus, there is no need to look further than probability for a language to talk about chance and likelihood. Furthermore, probability is a normative theory of reasoning under uncertainty. Its close cousin decision theory makes clear prescriptions about optimal decisions under uncertainty.

Our initial research in probabilistic temporal reasoning is covered in Chapter 2. We present a number of models of probabilistic temporal inference that we have developed. These include continuous-time models related to queueing theory and a discrete-time model related to the theory of Markov processes.

Our models emphasize reasoning about causation and the persistence of states in the world. These issues essentially remain open problems for commonsense reasoning in artificial intelligence. Our results have borne out the promise of applying probability theory, by offering viable solutions to issues that remain problematic in this area.

We can argue at length about the utility of a theory or approach to automated intelligence that employs an explicit, declarative representation of probability, especially as related to time. Instead, here we assume that such a representation is

interesting in and of itself, given the extent to which people talk about chances and probabilities. We assume that people would find it useful to have a database or simulator to which we can make queries about time and probability. The question then of whether or not they are useful for 10kg autonomous mobile robots on Mars is really an empirical one.

Logic of Lifetimes

The approach of reasoning about time, change, and chance with probability is given a formal grounding with the design of formal languages for writing down knowledge about time and probability. These are a family of language known as the *logic of lifetimes*. The logic of lifetimes is covered in Chapter 3. The logic of lifetimes takes some of the ideas developed in Chapter 2 and formalizes them as in the context of temporal logic. We formalize notions of facts, events, time, and probability, and provide syntax and semantics for successively more expressive languages for writing down knowledge about when facts are likely to be true, how long they persist for, and how their probabilities change over time.

Time Nets

The languages of Chapter 3 enable us to write down our knowledge about time and probability. We still need a method for making assessments about probability. This is done with with *time nets*. The time net is itself a *Bayesian network*, an augmented graph data structure for reasoning about probability. The time net is a Bayesian network that incorporates special features for temporal reasoning.

Research in the two disciplines of probability and decision theory has led us to understand the advantages of representing our knowledge as graph models. Bayesian networks and *influence diagrams* are the most commonly used of these models. The study of graphical dependency models for probabilistic theories is undergoing rapid advancement. We have exploited these advances to great effect in our research. In particular, algorithmic advances have enabled us to demonstrate feasibility of very fast probabilistic and decision theoretic computation in networks on the order of thousands of nodes. Computation in time nets, and in particular, the algorithms that we use, the clique tree propagation algorithm, and stochastic sampling algorithm, are discussed in Chapter 4.

Goo

We expect to see increasing need for *temporal information management* in extending the application of intelligent systems technology. As part of the thesis, we have developed a family of programs intended to serve as experimental testbeds for time management technology. These include `Sam`, which is a time net reasoner, and `Goo`, which is a *temporal database* built on top of a time net reasoner. Our focus is on decision support applications and on standalone embedded applications. Chapter 5 discusses the development of these implementations and experiments.

The temporal database, in our case a *probabilistic* temporal database, is a database for recording and keeping track of information about time. A user enters information about known facts and expected facts, and a set of rules about how facts interact to a temporal database. On the basis of this information, the temporal database tries to form a picture of what is likely to happen when.

For instance, a trucking company may wish to keep track of where its trucks are over time. The information given to a temporal database may be where trucks are known to be at what time, what time they leave known locations, and knowledge about how long it typically takes to travel on different routes. From such information, the temporal database tries to give users a picture of what is likely to be true over time.

There are other examples where a temporal database may be useful. A modern automobile assembly plant stocks a very small inventory of parts, and must keep track of which parts are needed and how long it will take to order and have them delivered. An intelligent agent in a distributed network may need to keep track of what activity is being undertaken in which part of the network. Another good example might be managing a hospital's physical and human resources.

In this thesis, we make a distinction between a temporal database and a temporal information management system. Roughly speaking, the job of a temporal database is to keep track of information. A temporal information management system is anything that uses a temporal database and the information stored or computed by a temporal database. For example, we may wish to keep track of the likely arrival time of all trucks, and be notified whenever any truck is more than 50% likely to fail to meet its deadline. The ability to predict the probability is handled by a temporal database. Keeping track of various probabilities is the job of a temporal information management system.

A temporal information management system may be used for decision support applications, or it may be used in standalone embedded applications. A temporal information management system in a decision support role may be constantly interacting with human operators or decision makers, providing information, making predictions, soliciting decisions. Or it may operate completely independently in managing a mobile robot or a manufacturing process. There is, of course, a whole spectrum in between these two types of operation.

The issues that arise in building such systems are not unique to the temporal information management domain. They cut across all of artificial intelligence, and especially all of representation and reasoning. As we shall see, `Goo` is a small step, one answer towards building capable time management systems.

The focus of the `Goo` experiment is on the basic support function of the temporal database. We build some simple temporal information management systems, in the form of planners based on decision theory, but the chief contribution is in the more basic temporal database technology. The development of the temporal database hinges on a combination of the two types of languages, logic and graphs, that we present in Chapters 2 and 4. We focus especially on the issues of time, change, and uncertainty, and the ability to change with time.

Making Plans and Decisions

Chapter 6 outlines the use of this research in making plans and in making decisions. A principal motivation for moving to a model of the world based on probability is decision theory. Decision theory is a normative theory of choice. In decision theory, choices are based on expectations of the value of an agent's actions. It is so closely related to probability theory that one cannot speak of one without the other.

Both the languages that we develop and the graph models that we develop are easily extended to represent decision making by decision theory. We show how to do this, and some examples of its use.

Chapter 2

Time and Chance

In this chapter, we introduce the topic of reasoning about change. Our interest is in the ability to reason about how things change over time and the chances that things may change in different possible ways. In this chapter, we focus on the representation of these entities with probability to numerically capture our notion of chance.

A key interest is issues of consistency, intuitiveness, and expressivity, as well as in forging links to past work in artificial intelligence. Issues of inference and computation will not be explored in depth here, being instead the province of later chapters.

First, we explore the idea of reasoning about change. Then we consider the explicit representation of uncertainty in the form of probability.

2.1 Reasoning About Change

Reasoning about change refers to activity that tries to capture the way in which the world behaves, such that it is possible to form ideas about what has happened or what will happen. A simple example is to try to predict the effects of administering some drug to a patient. How severe is the patient's condition likely to be 2 hours from now, 6 hours from now, or 2 days from now, if we supply the patient with Miraculo-17 now? What if we do not? What if we delay administering the drug, instead waiting for the result of some test, which is expected to arrive between 1 and 3 hours from now?

In a broad sense, reasoning about change may involve not just "thinking" but physical actions as well. This is especially true when we are learning about a situation at the same time that we are trying to reason about it. For instance, if we are trying

to predict how some mechanism, such as a bicycle part, will work under certain conditions, one easy method is to fiddle with the mechanism and see how it works. Although it could be argued that the reasoning in such behavior is separate from the concrete actions, it is nevertheless true that in the actual forming of conclusions, the physical acts are inseparable from, and no less important, than the thinking actions. Although we focus rather exclusively on more abstract aspects of reasoning about change, it is well worth remembering that it is but part of a larger whole.

In the typical kind of reasoning task that we are concerned with, we know some facts and some ways in which things typically behave and interact. Our task is to figure out what is likely to happen given that knowledge. Thus, our focus is on prediction. For the most part, we do not address the related problem of explanation, or figuring out what has already happened.[1]

As an example, we may try to estimate the arrival time of a spouse's airplane flight from Los Angeles. You know the scheduled departure time of the flight, and how long flights typically take to fly from Los Angeles. From that information, you will be able to predict what time the flight arrives.

Another example is that of a dispatcher at an automated warehouse who must decide what tasks are to be performed at what time [Dean and Kanazawa, 1989b]. At the warehouse, there are trucks delivering and picking up cargo all day long. Loading and unloading trucks is the job of automated robot handlers. Typically, the warehouse is so busy that not everything gets done as expeditely as one might hope. Trucks arrive with cargo, and no robot handlers are free to unload them, causing truck drivers to become irritated. Similarly, trucks arrive empty, expecting to load a shipment, but it is impossible to comply because a robot handler has just broken down and been sent back to the shop. Often, truck drivers become so impatient that they just leave to head to their next destination. Our task as the dispatcher is to assess how long each truck driver might stick around, so that with high probability, we load all the trucks, or failing that, those with high priority cargo.

So assume that our friend Sally the truck driver arrives sometime in the afternoon, expecting to pick up some cargo. Given that we know that she has arrived, what do we know about her chances of staying in the loading dock? How long, on average, can we expect her to hang around? Assuming that we are familiar enough with her behavior to know how impatient she is, we should be able to answer such questions

[1] We often find explanations during the process of prediction.

easily. There are various factors that may change our expectation about when she will leave. First of all, no matter what, she will leave by 4:30 so she can finish work by 5:00. However, even if she is feeling very impatient, if there is a coffee break, then she will probably stay a little longer to have some coffee and donuts, at least for the duration of the coffee break. Finally, if her friend Harry, another truck driver, arrives, then she will be likely to delay her departure for as long as he is there.

What can we say about such reasoning tasks? We identify four principal factors associated with this type of reasoning.

Time. First of all, reasoning about change inevitably involves the notion of time. We have seen that already in our example about Sally. To take another example, "after I turn the ignition key, the car will start" may or may not be an act of faith depending on the car, but either way, it involves a notion of time passing. Yet another example, "a few minutes after I turn on the rear window defogger, the rear window will gradually become clear" even more directly incorporates the idea of time. So without time, there would be no change to speak of, and perhaps vice versa. Without delving into any philosophical debate about the nature of time and change, we take it as given that time is central to reasoning about change, and that reasoning about change is synonymous with reasoning about time.

Causation. Another thing to notice about these last two examples is that inasmuch as they involve the notion of time, they also involve the notion of causality. A more accurate statement of our commonsense notion about turning the ignition key might be that "*only after* I turn the ignition, the car will start". So reasoning about change involves identifying and reasoning about causes and effects. This is not surprising given the close connection that is often made between time and causality. An example of this relationship is a common view of causality in terms of time, that "causes precede effects". In this view, turning the ignition *causes* the car to start, which is in accordance with the commonsense view. It would be mighty odd, after all, to have the key turned after the car started.

Incompleteness. A third facet of reasoning about change is that our knowledge is generally incomplete. We simply don't know everything, or perhaps much at all, except in some limited cases. In the medical domain, we don't know all the facts and relevant rules that pertain to a given situation. Indeed, it is rare that we know for certain exactly what the situation is. Another example of a type common to artificial intelligence is that you don't know if somebody has put a banana in the tailpipe of

your car, and therefore, that it is necessary to qualify your inference that if you turn the ignition key the car will start. All this does not mean that we cannot draw useful inferences on the basis of what we know. But it serves to illuminate the task that we have set ourselves. Furthermore, it lays the basis for accounts of change and of action that allow for reflection, *i.e.*, the ability to say something like "I don't know what time it is", and for actions designed to acquire knowledge. An example of the latter might be "if I call the supermarket, then I can find out if Ben and Jerry's ice cream is on sale".

Chance. Last but not least, there is always the element of chance at work in the world. We cannot reliably predict everything accurately ahead of time. The Hurricane Medusa heading northeastward may or may not hit your home town (of course, if your home town happens to be in Marin County, then the likelihood that it will hit is virtually nil). There is always the odd chance that an ATM machine will reject your bank card for no apparent reason. And we simply cannot know ahead of time how long Sally will stick around. Thus, in reasoning about change, we must take into account the fact that things do not progress in an orderly if-then fashion that enables us to predict the future with absolute certainty. We may only know how things behave on average.

The last two points often come under the single label of uncertainty, both in artificial intelligence and in philosophy. For example, in the *Bayesian* view [Savage, 1954], uncertainty and therefore chance is a *measure* of ignorance. If we had perfect information about the universe, then there would be no uncertainty, and thus no chance. We will not go into depth into this distinction here.

In the following discussion, instead of trying to capture reasoning in all forms, we typically ask ourselves to solve problems in some restricted domain. A domain might be the weather, driving a car, the warehouse, or even everything that concerns a particular autonomous mobile robot, although in practice, even those are large domains to capture completely with present artificial intelligence techniques. Rather, we are concerned with some subset of such a domain. We know some facts that pertain to such a subset of a domain. This might be the fact that it's currently December, and that it's summer in New Zealand. It might also include the fact that there are more sheep in New Zealand than there are people. At any rate, such assertions as we know about a domain will be called the *basic facts*. To make inferences about a domain, we also need some rules about the domain, what is sometime called the

physics of a domain. This might include such rules such as "if it's summer, then it's warm", and "if it's warm then warm-blooded animals feel warm". We call these rules the *domain theory* or the *causal theory*. Loosely, we may speak of a domain theory as the rules or the laws (of a domain). On the basis of the basic facts and the domain theory, we may make inferences such as that in New Zealand, the sheep are currently warm (of course, it's quite likely that sheep are always warm, but that's a different matter entirely). Such facts are *derived facts*, facts that are linked to basic facts or to other derived facts by rules in the domain theory.

A domain theory need not be the most faithful model of the world that one has at her disposal. It is conceivable that approximate models are adequate in certain situations. Intuitively, the more detailed and accurate a model, the more expensive it would be to compute with. This is a research area of growing interest in artificial intelligence, especially as pertaining to real-time systems. We have made preliminary investigations in [Kanazawa and Dean, 1989].

At any rate, we are interested in domains that contain basic facts about time, and domain theories about how facts and their chances change over time. Before we present our approach, let us first review past work in reasoning about change, emphasizing the development of temporal reasoning in artificial intelligence.

2.2 Temporal Reasoning in AI

In this section, we trace some of the major ideas in temporal reasoning in artificial intelligence. We note what the concerns were, and where appropriate, we note how they handle the issues of time, change, uncertainty, and incompleteness that we mentioned above.

2.2.1 The Situation Calculus

The most influential early work of reasoning about time and change is that developed in a seminal paper by McCarthy and Hayes [1969]. McCarthy and Hayes identified a number of areas that were seen as challenges in reasoning, one of which involved time and change. Here, they introduced ideas about what came to be known as the *situation calculus*.

The situation calculus was the first (AI) logical representation for reasoning about time and change. In the situation calculus the chief ontological entity is a set of

instantaneous states, called *situations*. The definition of a situation is [McCarthy and Hayes, 1969]:

> The complete state of the universe at an instant of time.

In practice, a situation is an abstract entity that is usually never fully specified. Instead, a situation is only partly specified by the facts that are known about it. Suppose that we are trying to describe our knowledge about Spaghetti cooking. Then for each situation, we might write down facts about water boiling, lighting the stove, and so on. However, we would probably omit mentioning facts about the number of planets or the genetic code of chimpanzees.

Each fact that we *do* mention about a situation is called a *fluent*. A fluent is traditionally expressed as a first-order predicate that associates a property with a situation. An example is `raining(`s`)`, which is intended to indicate that it is raining in situation s.

In the original treatment of situation calculus, in order to associate a property with an object in a situation, a function associating a fluent with the object was defined. For instance, to express that it is raining at location `x` in situation s, it was written as `raining(x)(`s`)`. In the example, `raining(x)` a function that returns a fluent indicating whether or not it is true in each situation s. Since a function returning a predicate is a second order construct, in subsequent treatment of the situation calculus this is normally written in one of two different forms.

One is the *reified* representation, which adopts a special predicate to associate the state of a fluent with a situation. Commonly, the predicate used is `holds`(*e.g.*, [Lifschitz, 1987]). In this approach, the previous is written as `holds(raining(x),` s`)` (or with the situation and fluent arguments reversed). The other approach associates the situation directly with each fluent. An example would be `raining(x,` s`)`.

The reified approach is perceived to have some advantages, the most notable being that it is easy to define special sentence formers convenient for use in reasoning about causes and change. For instance, `cause(raining, wet)` might be used to indicate that the fluent rain causes the fluent of wetness to become true. For this, and other reasons that we mention later, we adopt the reified approach from now on.[2]

To continue, in the situation calculus, typically, we know some facts about a situation, expressed in terms of fluents. Our interest is in knowing how situations

[2] Bacchus and others [1989] argue that a non-reified temporal logic is just as expressive as a reified one.

are related to one another, how a situation *changes* to become another situation. In the situation calculus, an *action* is a function that maps a situation to a new situation. The changes that the action precipitates is given by the difference between the situations before and after the action takes place. For instance, the change induced by the action of drinking a Coke may be that a Coke glass was full in the original situation, and empty in the resulting situation. This is often written as a *causal rule* such as the following.

$$\texttt{holds(full(g),}\ s\texttt{)} \supset \texttt{holds(empty(g), do(drink(g),}\ s\texttt{))}$$

Normally, an action such as `drink` is represented in this fashion, in terms of the `do` function, rather than as a function of situations. In this scheme, $\texttt{do(}\alpha\texttt{,}\ s\texttt{)}$ is the situation that *results* from doing action α in situation s. It is also common to use the `result` function in place of `do`.

In the implication above, the left hand side is known as the action *precondition*, the fluents that must hold for the action to take place, and the fluent argument to `holds` on the right hand side is known as the *effect*, the fluent that results from the action. Note that the rule above is implicitly quantified. It is implicitly within the scope of a universal quantifier that applies the rule to all situations s. Alternatively, we may consider the rule to be an axiom scheme.

In trying to formalize commonsense notions of change through the situation calculus, it was discovered that the use of such causal rules, although seemingly straightforward, were in fact far from free of problems. Some of the difficulties encountered came to be very well-known, as the *frame problem*, the *qualification problem*, and the *ramification problem*.

The frame problem, roughly speaking, is the problem of how to avoid having to write down what *does not* change with each action. There are many definitions of the frame problem, but the above definition suffices for our purposes. In trying to write down situation calculus axioms about how to undertake tasks, it was discovered that it may be necessary to write down knowledge about what remains true in each situation [McCarthy and Hayes, 1969]. For instance, suppose that our task is to deliver a package `X` from city `A` to city `B`. Then

$$\texttt{holds(loc(X,A),}\ s\texttt{)} \supset \texttt{holds(loc(X,B), do(move(X,A,B),}\ s\texttt{))}$$

describes how the location of `X` changes as a result of a `move` action. Suppose now that there is another package `Y` in city `A`. What happens to `Y` in the situation that

results from the move? Unless we happen to take `Y` along on the same trip, we assume that `Y` remains in city `A` when the `move` action takes place. In the original situation calculus, to make this commonsense inference, it is necessary to add the *frame axiom*

$$\texttt{holds(loc(Y,A),}\ s\texttt{)} \supset \texttt{holds(loc(Y,A), do(move(X,A,B),}\ s\texttt{))}$$

which says that the location of `Y` remains the same when we move `X` from `A` to `B`. The *problem* with this is that it is impractical to write down frame axioms for *everything* that remains true across all situations and all actions. To jump ahead a bit, our work represents an approach to dealing with the frame problem with probability.

The qualification problem is the issue of exceptions to causal rules. In practice, in trying to apply causal rules, there are always exceptions, and exceptions to exceptions. There is a common example for illustrating the qualification problem, which goes something like this. This is the example about starting an automobile that we saw earlier. 1) If the ignition key is turned, then the car will start. 2) Except if there is a banana in the tailpipe, in which case the car will not start. 3) Except in case the car has dual exhausts, in which it *will* start. 4) Unless there is a banana in *both* tailpipes. 5) and so on .... The perceived problem here is that there is never an end to exceptions, and thus, to write down our knowledge involves an endless number of rules.

The ramification problem is the problem of how to know what facts *do* change without explicitly stating them in causal rules. For example, suppose that a bomb `B` is contained in package `X`. Is there an easy way to infer that the bomb moves wherever the package moves, without specifying it explicitly? In some sense, this is the opposite analogue of the frame problem.

These three problems turn out to be quite general, common to many formalizations of commonsense temporal and causal reasoning. They are difficult to overcome in a succinct domain independent manner. Much of subsequent research in formalizations of commonsense reasoning in AI has been concerned with these problems.

There are technical problems with the situation calculus approach itself. One problem is that in the situation calculus, nothing changes unless an action takes place. Since the only thing that relates a situation to a situation is an action, if no action is known to occur, then we remain in the same situation. Thus, the situation calculus has difficulty in handling exogenous events. Conversely, if we are to postulate any changes at all, for instance, the passage of a moment of time, we must write down all the axioms necessary to relate one situation to the next.

Another problem is that it is difficult to have simultaneous actions. What if two move actions, of moving package `X` from `A` to `B`, and of moving package `Z` from `C` to `D` occur in the same situation? It is not at all clear from the simple situation calculus that we have so far seen how such might be accomplished, at least in the case where there are separate causal rules for `X` and for `Z`.[3]

Despite its problems, the situation calculus has been tremendously influential. There is a great deal of simplicity present in the approach, of perceiving the world in terms of distinct states and how they are related. It led in part to the STRIPS work described next, and later on to various attempts at *nonmonotonic reasoning* described later.

2.2.2 STRIPS

STRIPS was an early *planner* developed by Fikes and Nilsson [1971]. STRIPS was based on the situation calculus. In STRIPS, the world is represented as a database of known facts. Again, changes in the world occur on the basis of actions, according to what is specified in the action's *precondition-list*, *add-list*, and *delete-lists*. For instance, for the action of moving a package, `move(?agent, ?pkg, ?locA, ?locB)`, where `?X` is a *variable*, the precondition-list might be

```
loc(?agent, ?locA), loc(?pkg, ?locA),
```

the add-list

```
loc(?agent, ?locB), loc(?pkg, ?locB),
```

and the delete-list

```
loc(?agent, ?locA), loc(?pkg, ?locA).
```

STRIPS maintains knowledge about the state of the world on the basis of these lists. STRIPS may undertake an action if the facts in the action's precondition-list are true (exist in the STRIPS database). If this is the case, then it deletes the facts in the delete-list, and adds those in the add-list.

Thus, as far as STRIPS is concerned, a fact is true (remains in the database) as long as it is added at some time, and no action takes place that deletes it. This assumption that a fact remain true as long as no action takes place that specifies a

[3]It turns out that this can be accomplished [Schubert, 1989], but this result is tangential to this thesis.

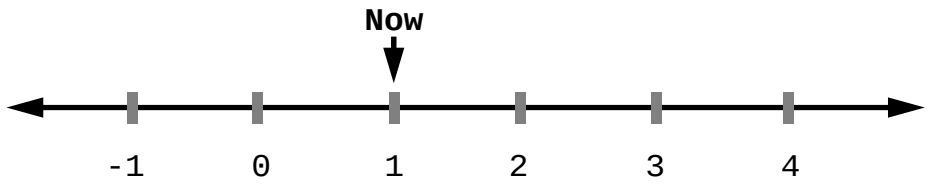


Figure 2.1: A (Discrete) Time Line

change in the fact's status is known as the *STRIPS assumption*, and it represents one approach to a *domain independent* way of attacking the frame problem. This assumption is present in many forms of temporal reasoning as we note later. STRIPS was also immensely influential, but it shares problems that the situation calculus has, such as difficulty with simultaneous actions. Thus, in planning, it is often supplanted by approaches that explicitly represent time.

2.2.3 Representing Time Explicitly

The early approaches to reasoning about time and change in AI such as the situation calculus and STRIPS are *state-based*. They associate a fact with a situation, and do not represent time directly. By contrast, much subsequent work on temporal and causal reasoning associates facts with time directly, either in terms of time instants or in terms of time intervals. In the rest of this thesis, we are mostly concerned with such explicit temporal representations.

In an explicit temporal representation, there is a totally ordered set or field of time points. Time points are conveniently represented as mapping to a number line. Such a line is a *time line* (Figure 2.1). If the time line is isomorphic to the integers, then the model of time is *discrete*. Otherwise, the time line maps to the real number field, and the model of time is *continuous*.

Technically, a key difference between the two models of time is that in the discrete-time case, there is a well defined notion of "the next time point", or "the next instant of time". The duration between successive time points may be constant or arbitrary, and as small or as large as is desired.

By contrast, there is no concept of the next instant in the continuous-time case. Another way to look at the difference is that in the continuous-time case, (given that it maps to the reals) it is possible to have a sequence of an infinite number of time points such that they follow one another between any two points of time u and v.

In our discussion below, we assume the existence of a time line. Therefore, time is totally ordered and non-circular. Normally, time is also infinite in extent both forward and backward. Furthermore, arithmetic involving time is assumed to be well defined.

McDermott was perhaps the first researcher in AI to adopt an explicitly temporal representation [1982]. In his paper, he defines a *temporal logic*, a logic for reasoning about time. The approach that we adopt in this thesis inherits much from what he develops there. In his logic, a fact is associated with time with the `T` operator, which stands for 'true'. For example, `T(s, loc(package-X, city-A))` says that package `X` is in city `A` in state `s`.

McDermott speaks about *states* such as `s`, rather than time points explicitly: each state has a *date* at which it occurs. Each date is a real number, and therefore, his representation is continuous-time. We can consider each state to be a time point with no loss of generality, and we adopt the notation `holds(t, loc(package-X, city-A))` instead, where `t` is assumed to be the date of state `s`. [4]

In addition to states, McDermott defines the *chronicle*, which is a collection of states. In his own words [1982],

> A *chronicle* is a complete possible history of the universe, a totally ordered set of states extending infinitely in time.

A chronicle is thus, roughly speaking, an extension of the situation of the situation calculus, a cross product of the time line with situations.

In fact, a chronicle is related to the idea of *possible worlds*. A possible world is an assignment of values true or false to all possible propositions in some theory. We may regard each situation, a 'slice' of the world at an instant as a possible world. A chronicle is the cross product of possible worlds with the time line.

With chronicles, the structure of time is *branching*. There is a tree of chronicles related by common facts. For example, suppose that we are driving through Manhattan. The set of chronicles may branch according to whether we take Broadway at Times Square, or we take 7th Avenue at Times Square. The opposite of a branching model of time is *linear*. With a linear model of time, there is basically just one model of the world that is being considered – there are not multiple branches.

[4] Commonly, the position of the fluent argument to the `holds` predicate is reversed between situation calculus and temporal logics. Although it may be confusing at first, we adopt the common approach here.

A key contribution of McDermott's work is the introduction into artificial intelligence of the idea that a fact stands for the set of intervals over which it holds true. Suppose for instance that package `X` is in city `A` from exactly `9am` to `3pm`. The way in which McDermott writes this would be `TT(s1, s2, loc(package-X, city-A))`, where `TT` stands for *True Throughout*, and where the date of `s1` is `9am` and the date of `s2` is `3pm` (we write this as `holds(9am, 3pm, loc(package-X, city-A))`). Then the *meaning* of the fact `loc(package-X, city-A)` is the time interval `[9am, 3pm]`. More precisely, what we associate each fact with is a set of interval - chronicle *pairs*. In other words, the above fact may be true from `9am` to `3pm` in one chronicle, from `10am` to `11am` in another, and not true at all in yet another.

McDermott makes a distinction between *facts* and *events*. Various researchers have tried to capture an intuitive notion of what a fact is, and what an event is. Generally, a fact corresponds to something like a fluent, whereas an event is something that "happens". Roughly speaking, a situation calculus action is something like an event. McDermott allows for events that take some duration of time to occur, rather than just an instant. In his language, an event is written with another special predicate, `occ`. For example, the event of "Cycling to work" from 8:15 to 8:30 in the morning might be written as

```
occ(8:15am, 8:30am, cycle-to-work)
```

The distinction between facts and events is arbitrary in some sense; it is possible to draw many different kinds of such distinctions, according to whether or not they stay true elastically (without interruption), and so on. Shoham gives an elaborate ontology of different types of facts and events based on such distinctions [1988]. As we see later, in this dissertation, we adopt a more specialized notion of fact and event for facilitating a semantics based on probability.

In either case, there *is* an important distinction that McDermott draws about events. This is the distinction between an *event type* and an *event token*. We indicated that a fact stands for the set of intervals over which it holds true. This is, in fact, true for events as well. Thus, the event "Cycling to work" may mean the interval `8:15am` to `8:30am` today. However, we probably do not go to work just one day, or perhaps even only once a day. In that case, there are *many* different intervals for which "Cycling to work" is true. In this case, we consider each uninterrupted interval for which an event is true as a *token*, and the entire set of tokens as the event *type*. This distinction is important in our integration of probability with temporal

reasoning.

Thus, in general, when we say something like `occ(1pm, 2pm, lunch)`, what we mean by `lunch` is really a token of the `lunch` type. As a symbol, it may really be an abbreviation for something like $\texttt{lunch}_{1001}$, the 1001st `lunch` token in our axioms. It is customary to take liberty with notation and let a type symbol serve as the token symbol except where the distinction is really crucial.

So how does an explicit temporal representation such as McDermott's differ from the situation calculus as far as representing change and the key problems mentioned before? A key concept that is important in the relationship between the two is the concept of *persistence*. A fact *persists* if it stays true over some period of time. A fluent that remains true over situations in the situation calculus is the same as a fact that persists in an explicit temporal representation.

To represent change in McDermott's approach, we generally use special axioms called `ecause` and `pcause`. The first stands for *event causation* and the second for *persistence causation*. As an example, to represent the previous example of the package moving between cities, we may write it as

$$\texttt{pcause(loc(X, A), move(X, A, B), loc(X, B)).} \tag{2.1}$$

The first argument to `pcause` is a precondition, the second argument is an event, and the last is the effect of the event, which is the beginning of a fact persistence. In some sense, the `pcause` rule above is similar in some respects to the causal rule

$$\texttt{holds(t, loc(X, A))} \wedge \texttt{occ(t, move(X, A, B))} \supset \texttt{holds(t +}\ \delta\texttt{, loc(X, B))} \tag{2.2}$$

where δ is a 'delay' between the action occurrence and the time when its effect becomes true.

So syntactically, a causal rule is somewhat similar in the explicit temporal representation and in the situation calculus. There are two important differences however. One is that in the situation calculus, an action completely specifies the difference between two situations. In a logic such as McDermott's, the effect of a causal rule only constrains the resulting state of the world; it does not specify it fully. The other difference is that in Equation 2.2, the asserted effect is that a fact is true at some point, whereas in Equation 2.1, the asserted effect is that a fact becomes true at some point, and begins to persist. This notion of a fact persisting involves some new machinery, that of *nonmonotonic reasoning*. This is covered in a little more detail

in the next section. The situation calculus notion of a causal rule is actually more similar in some sense to `ecause`.

Before we go on, we note that there is another influential approach to explicit temporal representation. Allen [1984] develops a temporal logic based on intervals, rather than points. In McDermott's approach, the interval is defined by two end points. Allen chooses the interval as the cognitively more plausible primitive, and represents time points as the intersection of intervals. Although there has been some debate about the merits of the two approaches, there do not appear to be substantial differences between them concerning their expressive and computational properties. Both approaches have made equal contributions in our understanding of the relationship between temporal reasoning and planning. We have chosen to discuss the *point calculus* approach in more detail, rather than the *interval calculus* approach. The reason is that it is more closely related syntactically and semantically to the approach that we develop Chapter 3, and because it was historically important in the development of nonmonotonic reasoning, which is described next.

2.2.4 Persistence and Nonmonotonic Reasoning

In work subsequent to the original situation calculus and STRIPS, it was widely perceived that classical logic is not suited to commonsense temporal and causal reasoning. In classical logic, if a fact is true, then adding further axioms cannot make the fact false again. This characteristic of classical logic is referred to as *monotonicity*. This is because asserted information is never retracted, and therefore the amount of facts that are known can only grow.

However, commonsense temporal and causal reasoning exhibits *nonmonotonic* behavior. In our commonsense view of the world, beliefs can be revised; that is, previously asserted facts can be retracted in light of new evidence. For instance, in the previous example of starting a car, if we don't know that a banana is in the tailpipe, then we assume that the car starts, and if we learn that a banana is in the tailpipe, then we revise our belief.

The key word here is 'assume'. In our commonsense reasoning, we make *default assumptions* about what is true and not true at will, and revise those assumptions as necessary. STRIPS made the assumption that facts stay true by default unless otherwise known to have become false.

Such a scheme is not within classic logic, and this has led to the desire to develop

nonmonotonic reasoning. The application of nonmonotonic reasoning is especially important in trying to solve the frame problem. A well-known attempt to solve the frame problem within the situation calculus is known as *circumscription*, due to McCarthy [McCarthy, 1986]. However, the first attempts at developing nonmonotonic reasoning was by McDermott again, in conjunction with Doyle [1980]. This approach was adopted in McDermott's temporal logic. Although the progress of the nonmonotonic reasoning enterprise is controversial to say the least, it is nevertheless quite central to the type of commonsense reasoning that we are interested in.

A key innovation in McDermott's logic is a means of specifying that a fluent persists by default, thereby 'solving' the frame problem. This is done via a special predicate `persist`. For instance, to note that the location of a package persists by default, we write `persist(loc(A, X))`.[5]

The meaning of `persist` is that a fluent persists by default, unless it is known that it ceases to be true, or is *clipped*. As with STRIPS, a fluent is not clipped unless it is *known* to be clipped. In McDermott's words

> Ceasing does not mean merely that the [sic] fact goes from true to false. In fact, ceasing is so rare that it never happens unless we hear about it.

The part about 'unless we hear about it' is formalized through a nonmonotonic assumption. The assumption is that a ceasing event does not occur 'as long as it is *consistent* that it does not occur': `consistent(nocease(?fluent))`.[6] A fact (or event) is consistent if it cannot be proven false. Thus, if no causal rule is present that predicts the occurrence of ceasing, then it is consistent that ceasing does not occur.

In [Dean and Kanazawa, 1988a], we called this assumption the *single default assumption of persistence*. Although this assumption, akin to a "law of inertia" seems to conveniently get around the frame problem by making it unnecessary to write down frame axioms, it presents other problems. The first is that it is now *necessary* to postulate a clipping event in order to infer that a fact becomes false. This is unrealistic at best. For example, suppose that we learn that a colleague in a city 300 miles away has a cold and roughly when the cold began. We may also know that a cold usually lasts about 3 days. However, unless we specifically learn that

[5]Actually, McDermott also gave an optional argument that specified the time *limit* for a persistence, but this does not actually alter the essential points that we raise.

[6]Note that McDermott actually used the operator `M` instead of `consistent`. We have chosen to use a more descriptive name.

the colleague became well, with the persistence assumption, we are forced to infer indefinitely that the cold persists.

Actually, as we noted earlier, in McDermott's original proposal, each `persist` was associated with a time limit. This limited the length of time that a fact would persist for by default. However, the selection of the time limit is itself an issue. For instance, suppose that half of all colds last 2 days and the other half lasts 6 days. What is the time limit to select in that case? Selecting one time limit tends to obscure the extra knowledge that we have. One better way is if somehow we could consider the *chance* that a cold, for instance, lasts for various lengths of time. This was in fact our initial motivation for considering the use of probability in temporal reasoning.

Perhaps a more damning problem with McDermott's approach concerns the general use of the nonmonotonic axioms such as involving `consistent`. An example of the problems with such approaches came to be very well known as the *Yale Shooting Problem* [Hanks and McDermott, 1987]. In this work, it was shown that a nonmonotonic logic such as McDermott's temporal logic could not draw some very simple commonsense inferences. It prompted a host of work that showed ways in which these inferences might be handled [Shoham, 1988; Haugh, 1987; Lifschitz, 1987].

The issues raised by the Yale Shooting Problem are important, as is much of the work that followed it. However, the technical details concerning this are somewhat tangential to this thesis. The main lesson that was drawn from the Yale Shooting Problem was that it is difficult to have a *domain independent* way of solving it. This was also another motivation for moving to a probabilistic or statistical approach to temporal and causal reasoning [Weber, 1989a].

2.3 Reasoning about Time and Probability

We will now explore the use of probability in temporal reasoning, outlining the development of our theory as a means of addressing the question of persistence. This dissertation is about an experiment in reasoning about time with probability. The motivation for using probability came first and foremost because of the problems with existing work as we described in the previous section. In order to make inferences about how long a fact remains true, we either had to write down highly domain specific and redundant frame axioms, or we had to assume that things stayed true forever unless it was otherwise really known to have become false.

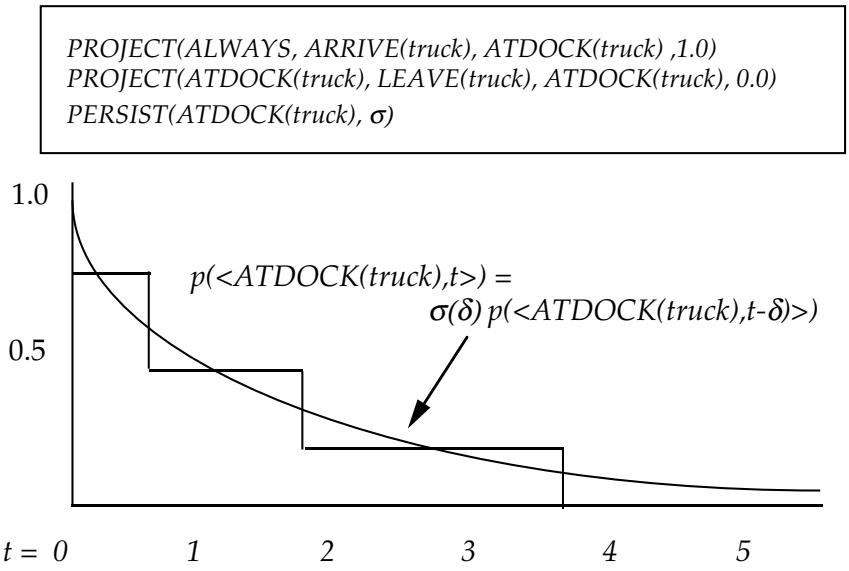


Figure 2.2: A simple domain theory

There is a growing body of work that addresses the problem of reasoning about time and probability within artificial intelligence. In this section, we review our own past work in the area.

2.3.1 ODDS

Our initial effort in probabilistic temporal reasoning and decision theoretic control was the ODDS project [Dean and Kanazawa, 1987; Dean and Kanazawa, 1989b]. ODDS was designed as a planning system incorporating a temporal database that applied *probabilistic causal rules* to aid the process of reasoning about actions. It was implemented as a prototype in a simulated manufacturing domain. As far as we know, ODDS was the earliest AI research effort in applying probabilistic inference to temporal reasoning, and in employing decision theory in planning involving time.

ODDS incorporated a continuous model of time. In ODDS our belief in the likelihood of facts and events is represented in terms of continuous probability density functions and probability mass functions. The actual implementation made discrete approximations to the continuous model.

ODDS emphasized prediction that involves reasoning about persistence. In a process called *probabilistic projection*, we provided a decision procedure with a polynomial time algorithm for determining the probability of the possible consequences of a set of events and initial conditions. The integration of simple probability theory with temporal projection enabled us to circumvent shortcomings that nonmonotonic temporal reasoning schemes have in dealing with persistence.

The ODDS system was organized into two components, a planner and a *probabilistic temporal database*. The probabilistic temporal database maintains a picture of the world. It keeps track of facts and events both observed and predicted, and it uses probabilistic projection to form expectations about what things are likely to become true. In addition, the probabilistic temporal database also refines the causal rules that it uses in probabilistic projection.

Let us present an example domain theory. Figure 2.2 depicts this simple domain theory graphically. This shows some rules for reasoning about the warehouse domain. Let τ stand for `?truck`, a *variable* that is assumed to be a truck, and let ξ stand for `Warehouse12`.

`project(always, arrive(`τ`, `ξ`), loc(`τ`, `ξ`), 1.0)`

`project(loc(`τ`, `ξ`), leave(`τ`, `ξ`), loc(`τ`, `ξ`), 0.0)`

`persist(loc(`τ`, `ξ`), `σ`)`

The first type of rule is a *projection rule*. The first rule states that following the arrival of a truck at `Warehouse12`, its location can be inferred to be the warehouse with probability 1.0. `always` is a proposition that is always true; hence its name. It means that the rule applies in every situation. The second rule says that if a truck leaves `Warehouse12`, then the probability that it's there is 0.0. The second type of rule is a *persistence rule*. Within an ODDS domain theory, we replace the single default rule of persistence described in Section 2.2.4 with a set of probabilistic rules: one or more for each fluent. Each persistence rule has the form `persist(`φ`,`σ`)`, where φ is a fluent and σ is a function of time referred to as a *survivor function* [Syski, 1979]. A survivor function tells us the probability that a fact φ remains true until at least time t, given that it became true at some initial time t_0:

$$\sigma_\varphi(t - t_0) = \texttt{P(holds}(t_0, t, \varphi) \mid \texttt{occ}(t_0, \texttt{beg}(\varphi))) \tag{2.3}$$

In ODDS, we simplify the model by considering only two types of survivor functions: exponential decay functions and piecewise linear functions. Formal analyses and

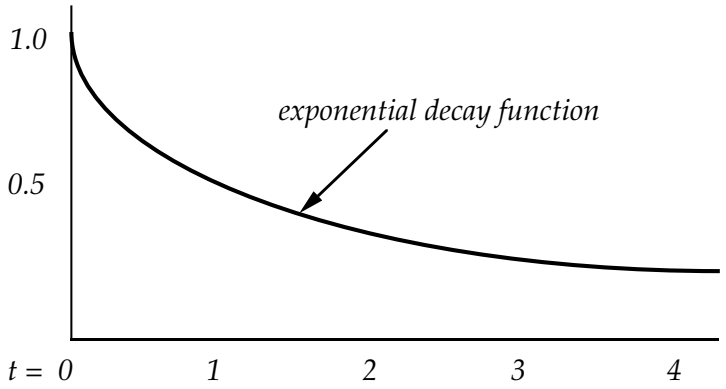


Figure 2.3: An exponential survivor function

proofs pertaining to the use of these functions are contained in [Dean and Kanazawa, 1989b]. An exponential decay function takes the form $e^{-\lambda t}$ where λ is the constant of decay. As an example, such a function may indicate that the probability of a truck remaining at the dock drops off as a function of the initial time t_0 since the truck arrived at an exponential rate determined by λ (Figure 2.3):

$$\texttt{P(holds(}t\texttt{, }\varphi\texttt{))} = e^{-\lambda(t-t_0)} \tag{2.4}$$

A persistence rule with an exponential decay function is usually simply written as `persist(`φ`,`λ`)`. The exponential decay function is well known for its *memoryless* quality. It is insensitive to changes in the time of occurrence of events that cause a fact to become true, and hence it is easy to compute efficiently.

We can generalize (2.4) and obtain:

$$\texttt{P(holds(}t\texttt{, }\varphi\texttt{))} = e^{-\lambda(t-t')}\texttt{P(holds(}t'\texttt{,}\varphi\texttt{))} \tag{2.5}$$

Equation 2.4 is a special case of Equation 2.5 where t' is the time that φ became true (thus `P(holds(`t'`, `φ`)) = 1.0`).

A survivor function of this form indicates how likely it is for a fact φ to stay true for various different durations without regard to the actual reason that eventually makes it false. For instance, a truck driver may leave because her truck has been loaded, or because she becomes impatient, or because it is time to go home. A survivor function makes it possible to reason about when a fact becomes false without regard to such specifics. It is sufficient to know how long it has passed since it has

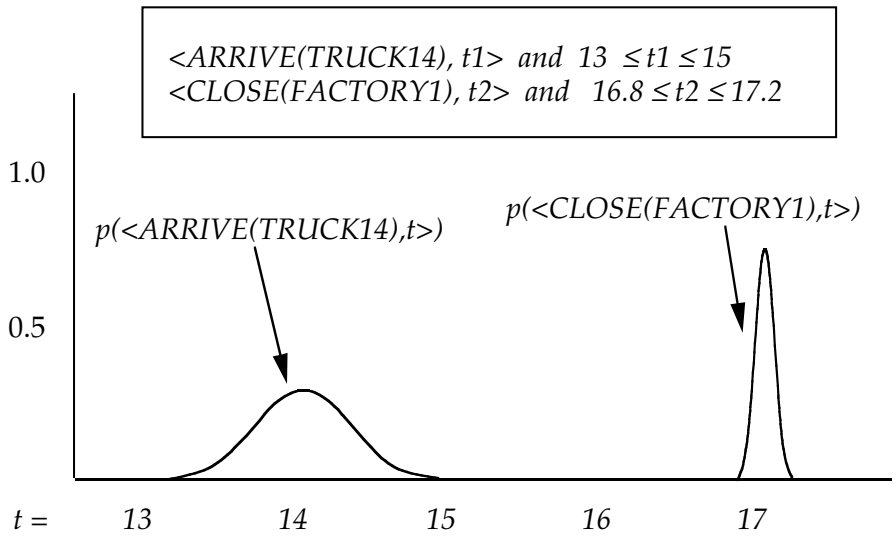


Figure 2.4: Some basic facts

become true to predict what the likelihood is that the fact is still true. In [Dean and Kanazawa, 1989b], we show how to construct a survivor function for a fact type φ from observations of when instances of φ become true and false.

Figure 2.4 shows a set of basic facts corresponding to two events assumed in our example to occur with probability 1.0 within the indicated intervals. `ODDS` assumes that there is a density describing the probability of each event occurring at various times, and uses a default density if no density is provided.

Evidence concerned with the occurrence of events and the persistence of facts is combined to obtain the density `P(holds(t,`φ`))` for a fact φ being true at various times by convolving[7] the probability `P(occ(z,beg(`φ`)))` with the survivor function σ_φ associated with φ:

$$\texttt{P(holds(}t,\varphi\texttt{))} = \int_{-\infty}^{t} \texttt{f(occ(}z,\texttt{beg(}\varphi\texttt{)))}\sigma_\varphi(t-z)dz \tag{2.6}$$

Figure 2.5 illustrates a simple instance of this kind of inference.

Equation 2.6 does not take into account the possibility of specific events known to make φ false. Suppose, for instance, that ε corresponds to Sally actually leaving the warehouse with `Truck22`. Then

$$\texttt{P(holds(}t,\varphi\texttt{))} = \int_{-\infty}^{t} \texttt{f(occ(}z,\texttt{beg(}\varphi\texttt{)))}\sigma_\varphi(t-z)\left[1-\int_{z}^{t}\texttt{f(occ(}x,\varepsilon\texttt{))}dx\right]dz \tag{2.7}$$

[7]The *convolution* of f with g is defined as convolved$f = \int_{-\infty}^{+\infty} f(x-u)g(u)du$.

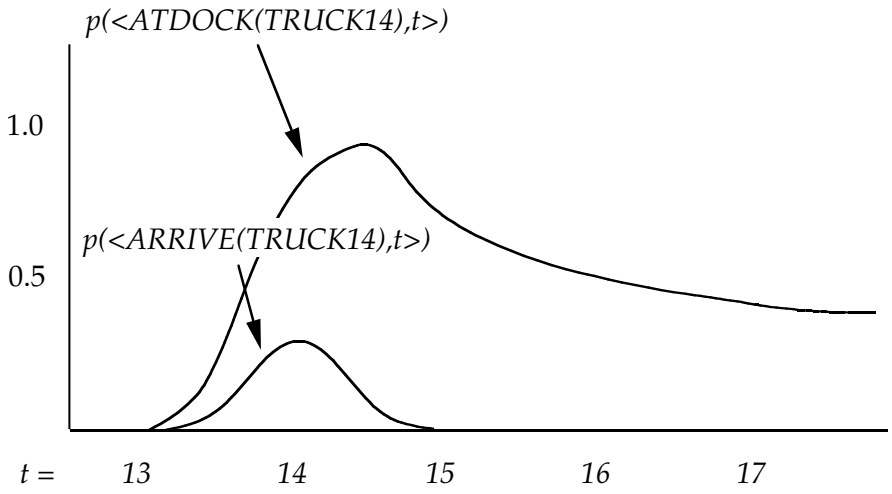


Figure 2.5: Computing probability by convolution

is more correct. Figure 2.6 illustrates the sort of inference licensed by (Equation 2.7). Note that the range of the resulting probability density is restricted; after the point in time labeled 17, the persistence of `loc(Truck22, Warehouse12)` is *clipped*, and thereafter its probability is represented by another density.

`ODDS` implemented a polynomial-time probabilistic projection algorithm. This was possible for two reasons. First of all, the restriction of survivor functions to exponential decay and piecewise linear functions turned out to be particularly expedient insofar as the evaluation of the convolution integral was concerned. In `ODDS`, the convolution integral was approximated as a Riemann sum. In the sum, we utilized a particular discretization of the time line into fixed sized intervals. The "memoryless" character of exponential decay function makes the summation process particularly simple. As it turns out, because of this memoryless character, it was possible to evaluate the integral in a single forward sweep through the time line. The same turned out to be true (with minor modifications) for the piecewise linear functions. The algorithmic details are presented in [Dean and Kanazawa, 1989b].

The second reason for the efficiency of the `ODDS` projection algorithm is that we made assumptions about facts in the database that precluded complex dependencies. In particular, we assumed that all antecedents to a projection rule were always conditionally independent. Thus, their joint probability is given by the product of their probabilities.

Unfortunately, we discovered that such an independence assumption rarely holds

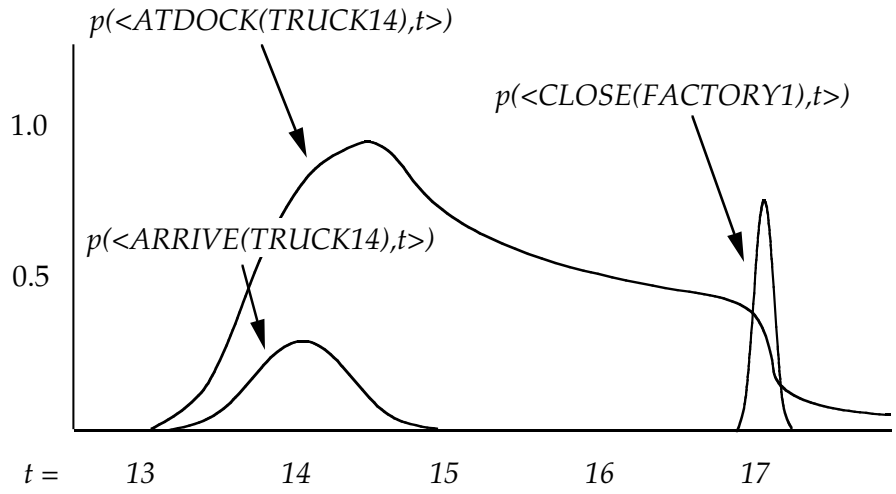


Figure 2.6: Probabilistic Projection

true for planning problems. Furthermore, there is a problem with equation Equation 2.7. The survivor function σ_φ was meant to account for all events that make φ false, but Equation 2.7 counts one such event, the truck leaving, twice: once in the survivor function and once in `f(occ(x,`ε`))`. In certain cases, this can lead to significant errors (*e.g.*, we always predict the truck leaving accurately).

2.3.2 A Discrete Approach

In order to remedy the problems with `ODDS`, we subsequently adopted a discrete model of reasoning about probability and time [Dean and Kanazawa, 1989a]. In this model, all factors contributing to our belief in the truth or falsehood of a fact or event are separated and made explicit. A discrete model of time is adopted. By a discrete model, we mean a state oriented model where we can speak about the existence of a time point, and a 'next' time point. The time line is divided into a finite *time partition* of points. The state of the world at each time point depends on the state of the world at other time points. In the simplest case, we let it depend only on the state of the world in the previous time point.

In the following, for a time point t, we loosely speak of its next time point $t+1$, or previous time point $t-1$. The difference between the time points depends on the discretization of time that is adopted, so '1' should be read as 'one time slice', whatever that may be. It is even possible that depending on what t is, the magnitude

of a time slice is different. In the discussion in this section, this is not a crucial point.

The survivor function of the previous section is replaced by conditional probability across time points. We let the probability of a fact φ being true at a point t depend on whether or not it was true at the previous point $t-1$:

$$\begin{aligned}\texttt{P(holds}(t,\varphi)\texttt{)} \;=\; & \texttt{P(holds}(t,\varphi)|\texttt{holds}(t-1,\varphi)\texttt{)} \times \texttt{P(holds}(t-1,\varphi)\texttt{)} + \\ & \texttt{P(holds}(t,\varphi)|\neg\texttt{holds}(t-1,\varphi)\texttt{)} \times \texttt{P}(\neg\texttt{holds}(t-1,\varphi)\texttt{)}\end{aligned}$$

where $\neg$holds(t,φ)$\equiv$ holds(t, $\neg\varphi$). The exponential decay survivor function is equivalent to the following case.

$$\begin{aligned}\texttt{P(holds}(t,\varphi)\mid\texttt{holds}(t-1,\varphi)\texttt{)} &= e^{-\lambda(\iota)} \\ \texttt{P(holds}(t,\varphi)\mid\neg\texttt{holds}(t-1,\varphi)\texttt{)} &= 0.\end{aligned}$$

Naturally, whether or not φ is true at time t is generally dependent on factors other than whether or not φ is true at time $t-1$. To combine the available evidence correctly, it helps to distinguish the different sorts of knowledge that are brought to bear on estimating whether or not φ is true. Let $\mathcal{E}(\varphi)$ and $\mathcal{C}(\varphi)$ respectively be events known to make φ true and false. We distinguish the different sorts of knowledge and evidence in terms of those events. This allows us to consider causation abstractly and uniformly for all propositions φ without regard to specific dependence of φ on other propositions. The following equation indicates how the evidence should be combined. Note that we assume that P(occ(t, $\mathcal{E}(\varphi)$) $\wedge$ occ(t, $\mathcal{C}(\varphi)$)) = 0.

$$\begin{aligned}&\texttt{P(holds}(t+1,\varphi)\texttt{)} = \qquad\qquad (2.8)\\ (N1)\quad & \texttt{P(holds}(t+1,\varphi)|\texttt{holds}(t,\varphi) \wedge \neg(\texttt{occ}(t,\mathcal{E}(\varphi)) \vee \texttt{occ}(t,\mathcal{C}(\varphi))))\\ & \quad \times \texttt{P(holds}(t,\varphi) \wedge \neg(\texttt{occ}(t,\mathcal{E}(\varphi)) \vee \texttt{occ}(t,\mathcal{C}(\varphi))))\\ (N2)\quad + \quad & \texttt{P(holds}(t+1,\varphi)|\texttt{holds}(t,\varphi) \wedge \texttt{occ}(t,\mathcal{E}(\varphi)))\\ & \quad \times \texttt{P(holds}(t,\varphi) \wedge \texttt{occ}(t,\mathcal{E}(\varphi)))\\ (N3)\quad + \quad & \texttt{P(holds}(t+1,\varphi)|\texttt{holds}(t,\varphi) \wedge \texttt{occ}(t,\mathcal{C}(\varphi)))\\ & \quad \times \texttt{P(holds}(t,\varphi) \wedge \texttt{occ}(t,\mathcal{C}(\varphi)))\\ (N4)\quad + \quad & \texttt{P(holds}(t+1,\varphi)|\neg\texttt{holds}(t,\varphi) \wedge \neg(\texttt{occ}(t,\mathcal{E}(\varphi)) \vee \texttt{occ}(t,\mathcal{C}(\varphi))))\\ & \quad \times \texttt{P}(\neg\texttt{holds}(t,\varphi) \wedge \neg(\texttt{occ}(t,\mathcal{E}(\varphi)) \vee \texttt{occ}(t,\mathcal{C}(\varphi))))\\ (N5)\quad + \quad & \texttt{P(holds}(t+1,\varphi)|\neg\texttt{holds}(t,\varphi) \wedge \texttt{occ}(t,\mathcal{E}(\varphi)))\\ & \quad \times \texttt{P}(\neg\texttt{holds}(t,\varphi) \wedge \texttt{occ}(t,\mathcal{E}(\varphi)))\end{aligned}$$

$$(N6) \quad + \quad \mathtt{P}(\mathtt{holds}(t+1,\varphi) | \neg\mathtt{holds}(t,\varphi) \wedge \mathtt{occ}(t,\mathcal{C}(\varphi))) \times \mathtt{P}(\neg\mathtt{holds}(t,\varphi) \wedge \mathtt{occ}(t,\mathcal{C}(\varphi)))$$

Consider the contribution of the individual terms corresponding to the conditional probabilities labeled (N1) through (N6) in Equation 2.8. (N1) accounts for *natural attrition*, the tendency for propositions to become false given no direct evidence of events known to affect φ. (N2) and (N5) account for *causal accretion*: accumulating evidence for φ due to events known to make φ true. (N2) and (N5) are generally 1. (N3) and (N6), on the other hand, are generally 0, since evidence of a fact that is known to make φ false does little to convince us that φ is true. Finally, (N4) accounts for *spontaneous causation*: the tendency for propositions to suddenly become true with no direct evidence of events known to affect φ.

If time is represented as the integers, we note that the law of inertia applies in those situations in which the terms (N1), (N2), and (N5) are always 1 and the other terms are always 0. For the rest of this section, we assume that time is discrete and linear and that the time separating any two consecutive time points is some constant δ. As before, we refer to consecutive time points as $t-1, t, t+1, \ldots$. Only evidence concerning events *known* to make φ true is brought to bear on $\mathtt{P}(\mathtt{holds}(\mathtt{t},\varphi))$.

Before we consider the issues involved in making predictions using knowledge concerning (N1) through (N6), we need to add to our theory some means of predicting additional events. We consider the case of one event causing another event. Deterministic theories of causation often use implication to model cause-and-effect relationships. For instance, to indicate that the occurrence of an event of type ε_1 at time t causes the occurrence of an event of type ε_2 following t by some $\delta > 0$ just in case the conjunction $\varphi_1 \wedge \varphi_2 \ldots \wedge \varphi_n$ holds at t, we might write

$$\mathtt{holds}(t, \varphi_1 \wedge \varphi_2 \ldots \wedge \varphi_n) \wedge \mathtt{occ}(t, \varepsilon_1) \supset \mathtt{occ}(t+\delta, \varepsilon_2).$$

If the caused event is of a type $\mathcal{E}(\varphi)$, this is often referred to as *persistence causation* [McDermott, 1982]. In our model, the conditional probability

$$\mathtt{P}(\mathtt{occ}(t+\delta, \varepsilon_2) \mid \mathtt{holds}(t, \varphi_1 \wedge \varphi_2 \ldots \wedge \varphi_n) \wedge \mathtt{occ}(t, \varepsilon_1)) = \rho$$

to indicate that, given an event of type ε_1 occurs at time t, and φ_1 through φ_n are true at t, an event of type ε_2 will occur following t by some $\delta > 0$ with probability ρ.

Let us take a slight modification of (2.8) as the basis for a model of persistence. (2.8) predicts $\mathtt{holds}(\mathtt{t},\varphi)$ on the basis of $\mathtt{holds}(t_0,\varphi)$, $\mathtt{occ}(\mathtt{t},\mathcal{E}(\varphi))$, and

occ(t,$\mathcal{C}(\varphi)$). In the model presented in this section, we only consider pairs of consecutive time points, and arrange things so that the value of a fluent at time t is completely determined by the state of the world at $t-1$. In (2.8), we interpret events of type $\mathcal{E}(\varphi)$ occurring at t as providing evidence for φ being true at t. In our new model, we interpret events of type ε_P occurring at t as providing evidence for φ being true at $t+1$. This reinterpretation is not strictly necessary, but we prefer it since the expressiveness of the resulting models can easily be characterized in terms of the properties of Markov processes. In our new model, we predict $A \equiv \texttt{holds}(t+1,\varphi)$ by conditioning on

$$\begin{aligned} C_1 &\equiv \texttt{holds}(t,\varphi) \\ C_2 &\equiv \texttt{occ}(t,\mathcal{E}(\varphi)) \\ C_3 &\equiv \texttt{occ}(t,\mathcal{C}(\varphi)) \end{aligned}$$

and specify a complete model for the persistence of φ as

$$P(A) = \sum P(A|C_1 \wedge C_2 \wedge C_3)P(C_1 \wedge C_2 \wedge C_3)$$

where the sum is over the eight possible truth assignments for the variables C_1, C_2, and C_3. Note that this model requires that we have probabilities of the form $p(A|C_1 \wedge C_2 \wedge C_3)$ and $p(C_1 \wedge C_2 \wedge C_3)$ for all possible valuations of the C_i.

In Section 4.3.1, we explore further how such a framework can be modeled using graphical dependency models.

2.4 Related Work

2.4.1 Temporal Reasoning

Temporal reasoning has traditionally been approached in AI in logical formalisms without explicit consideration of uncertainty as we noted in Section 2.2. Reasoning about time in the sense of explicit reasoning about time points and intervals is a comparatively later development in artificial intelligence, starting initially in the 1970s. In addition to the work already mentioned, some notable early work include that by Hendrix [1973] and by Kahn and Gorry [1977]. More recent work, focusing on applications in planning include Vere's work on DEVISER [Vere, 1983], Vilain [1982], and Dean [1985]. Rit [1986] and Dechter and Pearl [1989] show temporal constraint

propagation systems for efficient inference involving temporal intervals. Dean's TMM [Dean, 1985], a temporal database system, is covered further in Section 5.4.

2.4.2 Probabilistic Temporal Reasoning

Since we began investigations in reasoning about time and probability, there has been a small but growing body of work in this area in artificial intelligence.

Cooper, Horvitz, and Heckerman

Cooper, Horvitz, and Heckerman [1988] present a method for reasoning about the changing probability of a patient's condition over time. Apart from our own work in ODDS, this may be the earliest known work in probabilistic temporal reasoning in artificial intelligence. The approach taken in computing and reasoning about probability is similar to ODDS, in that the time line is partitioned into fixed length intervals, and then probability propagation is performed over those intervals.

Hanks

Hanks' work leading up to his dissertation [Hanks, 1988; Hanks, 1990] was another early example of work in probabilistic temporal reasoning contemporaneous with ODDS. Hanks' thesis makes two main contributions. First, his probabilistic temporal manager is capable of multiple granularities of time, for instance, reasoning over different time scales such as minutes and days. Second, he provides a heuristic algorithm for providing estimates about the probability of facts at particular time points. His algorithm is able to answer $P(\phi(t) > e)$ for some threshold e in an efficient manner. His algorithm is quite complex and relies on various special-case independence assumptions. Even simple questions such as conjunctive queries involving dependencies must be handled specially. He is unable to compute a consistent joint distribution. Depending on the query, there may be different probabilities assigned to a given proposition.

2.4.3 Mathematics

The present work leverages off mathematics in a fundamental manner. There is a comparatively long history of work in mathematics and statistics in the area of probabilistic inference with explicitn consideration of time. This area is generally known

as the study of *stochastic processes*, and spans a wide variety of different subareas, including queueing theory, Markov decision process theory, the study of survival analysis, and time-series analysis. The key distinction between work in mathematics and the current work is that we are concerned with issues in flexible representation and efficient inference for application in real problem solving domains, whereas work in mathematics often focus on the type of underlying theory on which we have built our work.

Operations Research

Some of the closest work in probabilistic temporal reasoning have come from operations research in the context of Markov decision process theory [Howard, 1960; Howard, 1969; Howard, 1971]. Our discrete-time time model of probability is essentially a Markov process. The fundamental results in this area come from the 1960s. It was impractical to attempt application of the ideas on the computing machinery of that time. Some of our present work, especially the experimental computational aspects outlined in Chapter 4 and Chapter 5 extends the practical applicability of the past work, both in computational and representational terms.

Survival Analysis

Another area of close relation to our present work is the work in survival analysis [Cox and Oakes, 1984]. Survival analysis is concerned with predicting how long a fact stays true in the world for. The chief domain is that of medicine, where duration of diseases and life expectancy are modeled. `ODDS`'s continuous-time model, with its use of survivor functions have a basis in this area. Similar ideas are used in queueing theory, which is concerned with expected time between events, and expected time waiting for events, especially in the domains of manufacturing, transport, and communications.

Time-Series Analysis

Time-series analysis [Box and Jenkins, 1976; Granger and Newbold, 1986] is concerned with temporal prediction of a series, and can be closely related to Markov processes. Times-series analysis has been applied in financial modeling and other areas. We have not explored the relationship of the current work with time-series analysis in great depth.

Chapter 3

The Logic of Lifetimes

In the last chapter, we showed different ways of expressing knowledge about time and probability and how such knowledge can be used to reason about the probability of facts and events over time. In this chapter, we show how to express the same knowledge in a set of formal languages called the *logic of lifetimes* (**LL**). The logic of lifetimes is a family of languages for expressing knowledge about time and probability. They are very closely related to the logic of time by McDermott [1982] that we showed in Section 2.2.3, as well as to its more modern incarnation in Shoham's [1988] logics of time. To those temporal logics, **LL** adds features for reasoning about probability in the spirit of the logics of probability by Bacchus [1991] and by Halpern [1989]. Through this exercise, we show that it is easy to incorporate a calculus of chance into the traditional AI account of commonsense reasoning. The resulting logics have the same advantages as the approaches of the previous chapter, with the further advantage of having formal semantics for the sentences in the language. Furthermore, as we show in the next chapter, it is easy to tie into efficient computation methods.

LL includes languages that are continuous-time and those that are discrete-time. All the **LL** languages that we define are temporally-quantified. In general, in describing change, it is convenient to have variables and quantified statements (*e.g.*, with 'for all', and 'there exists') concerning time. This is especially true for continuous-time logics of time and probability. For instance, it is difficult to consider how to have probability density functions of time without some form of temporal quantification.

It is likely that for certain applications, a language with only time constants is adequately expressive. This is especially true for the discrete-time case. Such a language may be useful for specifying the information in a *circuit* for reasoning about time and probability. We do not actually define such a language, although some of

the domain theories that we develop could be described by such a language.

The logic of lifetimes are three-sorted logics, with a sort that corresponds to each of the different kinds of things that we describe, domain objects, time, and probability. What this means is that the different type of objects are syntactically distinguished in sentences of the logic. The domain objects are the things of interest to us, such as packages or trucks, as well as predicates that are used to describe facts and events, such as moving packages or the light switch being on. Time and probability are basically self-explanatory.

If an **LL** has only constants and predicates for domain objects, then we say that it is propositional. Otherwise, (it has variables, quantification, and functions), it is first-order. The label of propositional and first-order with respect to **LL** only refers to the domain objects; **LL** is always first-order with respect to time, as we noted before. It may or may not be first-order with respect to probability, as we note later.

In this chapter, we show three of the **LL** languages. **LL1** and **LL2** are both continuous-time. The first has a model theory, whereas the latter, a more expressive language, does not yet. Thus, we call **LL2** merely a *language* rather than a logic. The other logic is **LLD**, which is discrete-time. All of these logics are propositional in the sense above. As it turns out, for our purposes, the first-order extensions do not substantially illuminate our understanding of reasoning about time and probability. In fact, it only complicates the model theory. Thus, we do not explore them at length here.

Before we present the languages, we list some of the basic commitments that we make as far as the kind of things that we wish to express and represent. We also clarify some of the terminology that is used in the definition of the logics.

3.1 Preliminaries

3.1.1 Basic Choices

Continuous-time or discrete-time?

A key decision in a logic of time is the structure of time itself, whether or not it is continuous or discrete. The basic underlying model of time of the logic of lifetimes is continuous, as with `ODDS`. However, we find it convenient to define sublanguages of **LL** that are more like the discrete-time model we saw in Section 2.3.2. These

sublanguages are special cases of the continuous-time case, where the time-line is broken up into intervals, and where assertions can only be made in terms of the intervals.

Such a discrete model of time is always at best an approximation. There is always an issue over how to handle events that fall "between" two consecutive time points. There is no good general solution for this. In the end, it comes down to a choice of which of the two time points to pick, perhaps both. This choice is domain dependent. If choosing one or the other leads to problems (for instance, if we cannot determine which of two events occur first because the approximation is too coarse), then it may be necessary to make the intervals between time points finer. That in turn may be an unacceptable burden for the model builder or user. There may be too many time points to represent easily, to keep track of, or to store rules for.

There is actually another, perhaps more natural, way to view the discrete-time case. Recall that in a discrete-time model, the time line is broken up into many intervals of time. The issues about events "falling between" time points arise only when the basic entities of time are the boundary points between the intervals. If the basic entities are the intervals themselves, then we do not have that problem (except now we potentially have the problem of events that fall *on* the boundary). This is in fact the view of discrete time that we choose.

In practice, the differences between the continuous-time and discrete-time approaches can be more philosophical than real. This is especially true when we consider in computation. A continuous-time model usually needs to be grounded out to some discretization at computation time.

Facts and Events

We make a distinction between fact and event in the logic of lifetimes. A *fact* is something that remains true for some time. It becomes true at a certain time and becomes false at a later time. The light being on in the office is a fact. It is essentially a fluent, and it *persists* as before.

An *event* is something that "happens" and then is no more. The light being turned on in your office is an event. Each event has a unique *date*, the point of time at which it occurs. The date of an event ε is written as $\Delta(\varepsilon)$.[1]

[1] An event may also *not* occur. In some cases, it is convenient to regard the date of an event that never occurs as positive infinity.

By contrast, a fact does not have a date, but a *range* over which it is true. The start point of the range is the event of the fact becoming true, and the finish point of the range is the event of the fact becoming false. We call the former the fact's *enabling event*, and the latter its *clipping event*. The enabling event of a fact φ is written `beg(`φ`)`, and its clipping event is written `end(`φ`)`. The range of a fact is `[`Δ`(beg(`φ`)), `Δ`(end(`φ`))]`.

Note that although a fact has a unique maximal *range*, it does not have a unique interval over which it holds true. It is true in any interval contained in the range. Whereas `occ(t, `ε`)` is true iff `t = `Δ`(`ε`)`, `holds(u, v, `φ`)` is true for any interval `[u, v]` such that Δ`(beg(`φ`)) ≤ u ≤ v ≤ `Δ`(end(`φ`))`. Thus the meaning of `occ` is not completely isomorphic to the meaning of `holds`.

In a sense, the event is more primitive compared to the fact because a fact is easily and naturally represented by the events of its becoming true and false.

Types and Tokens

As with McDermott's logic [1982], we make a distinction between fact (event) *types* and *tokens*. If a fact becomes true and false repeatedly, then each instance of the fact being true is represented as a separate token. The set of all tokens for a fact is the fact type. If an event occurs on more than one occasion, then each occasion of its becoming true is a separate token, and the set of all tokens for an event is the event type.

Thus, if a fact (token) φ holds over an interval, then it holds over all subintervals of that interval. Such a fact is called *liquid* by Shoham [1988].

The type/token distinction is particularly well motivated for a probability logic as we see later. Note that it *is* possible to consider the probability that *some* instance of a fact type is true at some point or interval of time. This would depend on the probabilities of the different possible instances. However, this is no longer a fact in a primitive sense: it implicitly involves quantification of some kind.

Associating facts, events, and time

We prefer a reified language for the following reasons. First of all, it is common to the logics with which the logic of lifetimes have the most affinity, such as McDermott's and Shoham's. Second of all, we will be defining logics that are temporally quantified but otherwise propositional. In such logics, the temporal arguments to the `holds` and

`occ` predicates can be quantified, but not the fact or event argument. While not an essential feature, the reified language is syntactically cleaner for this case.

What do we associate facts and events with? Since a fact persists, and an event is an instantaneous "happening", it is natural to associate a fact with an interval, and an event with a point of time. Thus, a fact is usually written in a sentence as `holds(u, v,` φ`)` where `u` and `v` are time points, and an event is usually in a sentence as `occ(u,` ε`)` where `u` is a time point.

For convenience, we allow the expression `holds(u,` φ`)` meaning that φ is true at time `u`. Furthermore, we allow `occ(u, v,` ε`)` to mean that ε occurred between the time points `u` and `v`. What we mean by that is

$$\exists t\ u \leq t \leq v\ \texttt{occ(t,}\ \varepsilon\texttt{)} \wedge \neg\exists t' u \leq t' \leq v \texttt{occ(t',}\ \varepsilon\texttt{)}$$

In other words, ε occurs at one point in the interval `[u, v]`, not over the entire interval.

Probability on Possible Worlds or on the Domain?

The main ideas behind logics with a subjective interpretation of probability is that there is a probability distribution over possible worlds. These logics directly take after Nilsson's *probabilistic logic* [Nilsson, 1986], as with the logics by Fagin and others [Fagin *et al.*, 1990], and by Halpern [Halpern, 1989]. Note though that theoretical computer scientists and philosophers predate the artificial intelligence practitioners with regard to this.

We adopt the possible worlds approach. The main reason is that the notion of possible worlds has a natural fit with temporal logics, and a number of temporal logics have possible worlds semantics.

3.1.2 The Object, Time, and Field Sorts

As we mentioned, the logic of lifetimes combine elements of Shoham's logics of time [1988] with Bacchus and Halpern's logics of probability [Bacchus, 1991; Halpern, 1989]. Basically, the features related to time are borrowed from Shoham, and features related to probability are borrowed from Bacchus and Halpern. Each of those logics have multiple sorts, Shoham's to represent objects and time separately, and Bacchus and Halpern's to represent objects and probability separately. Naturally, one way to

combine the two approaches is to consider logics with three sorts to represent objects, time, and probability separately. This is exactly what we do.

The three sorts in the logic of lifetimes correspond to each of the different types of things that we wish to represent, domain objects, time, and probability. These are the object, time, and field sorts. In general, a sort may contain constants, variables, functions, and relations. As we see later, our logics are inductively built up from terms of the different sorts, much as with standard propositional and first-order logic.

The object sort O is meant to represent objects of interest in the domain. In particular, the object sort is taken to be a countable collection of objects, including things like your neighbor Sally, the truck she drives at work, and the warehouse on Gravel Avenue where she often takes deliveries. Within the logics, there are object constants, having names like `Sally` or perhaps `Sally15`, `Truck22`, and `Warehouse12`, and in first order languages, object variables, having names like `driver`.

Typically, there are object functions and relations of various arities. As usual, it is possible to regard the constants in the language as functions of arity 0. Examples of object predicates might be `address(Warehouse12, GravelAvenue0)` and `female(Sally)`, intended to mean, respectively, that the address of the warehouse denoted by `Warehouse12` is Gravel Avenue and that Sally is female. It is often convenient to include the equality predicate = that returns true if and only if two objects are the same.

Examples of an object function might be `work-truck(Sally)`, which is presumed to denote the truck that Sally drives at work. As usual, we can just as easily represent the same information in an equivalent predicate format such as `work-truck(Sally, Truck22)` being true, assuming that the function would have denoted `Truck22`. This would allow us to remain in a propositional framework.

The time sort T is used for expressing time. All of our logics feature quantification with respect to time, and the time sort may contain constants, variables, functions, and relations.

We assume that the notion of dates and times are well-defined and that they map appropriately to the time line. For example, `2:30pm` is assumed to refer to 2:30 in the afternoon today, which might really be the number `152345828.0`. We define the distinguished time predicate $\prec$ that defines precedence between time points. Finally, we assume that arithmetic with time points is well defined. For example, we assume that it makes sense to say something like `t1 = 2:30pm + 30min`. Toward that end, we

include a number of standard infix mathematical functions among the time functions, such as `+`, `-`, `*`, and `/`. As described below, functions of the time sort, functions that map to the time sort, are allowed to have arguments of both time sort and field sort to simplify definitions of such arithmetic.

The last sort is the field sort `F`. The field sort is used to represent probability as well as numerical objects or quantities in general. The field which we map to is the real number line. In terms of probability, this means that we adopt a standard axiomatic definition of probability with a countably additive measure, a σ algebra. Note that this is similar to the conventions that Fagin, Halpern, and Megiddo adopt in their logics of probability [1990], as well as in Halpern's first-order logics of probability [1989]. Although their main discussion is not in terms of real-valued probability, but rather in terms of rational valued polynomial *weight formulas*, they also show how to extend to the case of real numbers [1990]. This is in contrast to Bacchus's approach, which adopts not the real line, but arbitrary totally ordered fields [Bacchus, 1991].

Again, the field sort contains constants, variables for the first-order case, functions, and relations. There are typically a very rich class of field functions, to represent probability density functions. It is virtually impossible to define any interesting probability densities without field functions. An example might be `neg-exponential(`λ`,` t`)`, which represents a function that has the negative exponential distribution with the parameter denoted by the field variable λ for time t. There are also functions that map from the object sort to the field sort. These are "measuring" functions like those defined by Bacchus [1991]. These measuring functions enable us to express statements about numerical quantities, like "the level of the water tank is 65 gallons". Within our logics, such a statement might be written as `level-gallons(tank17) = 65`. `level-gallons` is a measuring function, and `tank17` and `65` are constants of object and field sorts respectively. Again, we could remain in a propositional framework by representing the equivalent information in predicate form.

As we touched on earlier, functions that map to the time sort or the field sort may take arguments of both time and field sorts. This feature is provided for convenience in defining arithmetic and mathematical functions such as probability density functions of time. For instance, we can define functions such as `neg-exponential2(12:00pm,` λ`,` t`)`, which is assumed to be a function that returns the negative exponential distribution, for time t, starting with probability 1 at noon, and decaying according to the field variable λ. Also, allowing multi-sorted function arguments makes it legal to

write down arithmetic expressions such as `(4:00pm - 1:00pm) / 2`, where the first two numbers are time points, and `2` is a field constant.

3.2 LL1

We are now ready to present the first of our logics. **LL1** is a propositional version of **LL**.[2] It has the minimum essential features for representing probability in a continuous-time framework, such as we saw for `ODDS`. **LL1** is kept propositional to show how easy it is to build a useful continuous-time probability language, without obscuring it with extra features for handling quantification for domain objects.

In both this and the logics that follow, we assume the existence of some alphabet, or a set of symbols Φ for denoting objects, and various function symbols of fixed arity. As usual, we regard constants as functions of fixed arity, and predicates as functions with true/false value.

We begin presenting the syntax for **LL1** by defining the entities in each of the three sorts.

1. The object sort:
   - $\mathsf{O_C}$: A set of object constants.
   - $\mathsf{O_P}$: Object predicates of various fixed arity.
2. The time sort:
   - $\mathsf{T_C}$: A set of time constants.
   - $\mathsf{T_V}$: A set of time variables.
   - $\mathsf{T_U}$: $\mathsf{T_C} \cup \mathsf{T_V}$.
   - $\prec$: The time precedence operator.
3. The field sort:
   - $\mathsf{F_C}$: A set of field constants.
   - $\mathsf{F_f}$: Field functions of various fixed arity.
   - $=$: The field relation operator.

[2] A logic essentially similar to **LL1** was presented in [Kanazawa, 1991] under a different name.

We also have the probability operator P.

Recall the fact that we allow field functions over both time and field sorts. In the below, a $n(i,j)$-*ary function* is a function of arity n, with i time sort arguments and j field sort arguments, where $n = i + j$, and both i and j can be 0. Given the above, we define **LL1** inductively as follows.

First of all, the terms are defined as follows:

- A single object constant is an *O-term*. A single time variable or constant is a *T-term*. A single field constant is an *F-term*.
- Let g be an $n(i,j)$-ary field function symbol. If $t_1, \ldots, t_i$ are T-terms, and $t_{i+1}, \ldots, t_n$ are F-terms, then $g(t_1, \ldots, t_n)$ is an F-term.

DEFINITION 3.1
Let p be an n-ary object predicate symbol, and let $o_1, \ldots, o_n$ be O-terms. Then $p(o_1, \ldots, o_n)$ is an O-term. Furthermore, $p(o_1, \ldots, o_n)$ is a proposition. The set of all propositions is $\mathcal{P}$.

We assume that a distinction is made between facts and events within the propositions, where $\mathcal{P}$ consists of disjoint sets, the facts $\mathcal{F}$, and the events $\mathcal{E}$.

Given the above, we can define the well-formed formulas (wffs) as follows:

- If u and v are T-terms, then $u \prec v$ is a wff.
- If u and v are T-terms, and $\pi \in \mathcal{F}$, then $\texttt{holds}(u, v, \pi)$ is a wff.
- If u is a T-term, and $\varepsilon \in \mathcal{E}$, then $\texttt{occ}(u, \varepsilon)$ is a wff.
- If φ is a wff not containing P, and f is an F-term, then $\texttt{P}(\varphi) = f$ is a wff.
- If φ_1 and φ_2 are wffs, then so are $\varphi_1 \wedge \varphi_2$, and $\neg\varphi_1$.
- If φ is a wff and $u \in \mathsf{T}_\mathrm{V}$, then $\forall u \varphi$ is a wff.

Through standard definitions, we define $\vee$, $\rightarrow$, $\exists$ in terms of $\wedge$, $\neg$, and $\forall$. Similarly, we define an extended set of time relations, such as $=$[3] and $\in$, which denotes time interval membership. A time interval is denoted as `[u, v]` for two time points `u` and `v`. We also define a set of field relations similarly.

[3]We have chosen to use the same symbol to indicate both time equality and field equality. We assume that it is obvious by context which one is meant.

For convenience, we can also define the negated time and field relations, as well as standard abbreviations such as:

$$\begin{aligned} \forall x, y \quad &\equiv \quad \forall x \forall y \\ x \prec y \preceq z \quad &\equiv \quad (x \prec y) \wedge (y \preceq z) \\ &etc., \ldots \end{aligned}$$

Finally, we allow the abbreviations `holds(`t`, `π`)` as shorthand for `holds(`t`, `t`, `π`)`, and `occ(u, v, `ε`)` for

$$\exists t \; u \leq t \leq v \; \texttt{occ}(t, \varepsilon)$$

Conditional probability is defined with the following abbreviation.

$$\texttt{P}(\varphi \mid \psi) \triangleq \texttt{P}(\varphi \wedge \psi) \; / \; \texttt{P}(\psi)$$

where φ and ψ are wffs.

If a variable u appears in a wff with no quantifier naming the variable, then it is a *free variable*. A *sentence* is a wff without free variables. Some examples of sentences in **LL1** follow. First, let us consider some examples that show its function as a logic of time. As a simple exercise, we define the predicate `always` that is always true:

$$\forall u, v \; \texttt{holds}(u, \; v, \; \texttt{always})$$

To express that Ben and Jerry's is open from 10 in the morning to midnight, we would write:

```
holds(10am, 12am, open(Ben-and-Jerry's))
```

Next, let's consider **LL1** as a logic of time and probability. The predicate `always` above has the following property:

$$\forall u, v \; \texttt{P(holds(}u\texttt{,}v\texttt{,always))} = 1.0.$$

Survivor functions, for all time t after some starting time t_0 can be given by the survivor function σ: [4]

$$\forall t \; t_0 \preceq t$$
$$\texttt{P}(\neg \exists t_1 \; t_0 \preceq t_1 \prec t \wedge \neg \texttt{holds}(t\texttt{,}\pi) \mid \texttt{holds}(t_0\texttt{,}\pi)) = \sigma_\pi(t_0, t)$$

[4] We could have assumed that $\sigma(t) = 0$ for all $t < t_0$ instead of assuming that $t_0 \preceq t$

We need the survivor function to be in the above form rather than $\sigma_\pi(t-t_0)$, because strictly speaking, **LL1** lacks arithmetic between time points.

We can now also write down axioms about our friend Sally. To keep our axioms short, in the following, we omit the `Warehouse12` argument, and let `here(Sally)` stand for `loc(Sally, Warehouse12)`. Given this, we can write down the simple version as:

$$\forall u, v\ \texttt{P(occ(}u\texttt{, }v\texttt{, arrive(Sally)))} = \texttt{norm(2pm, 0:15, }u\texttt{, }v\texttt{)} \quad (3.1)$$

$$\forall t, t_0\ \texttt{P(}\neg\exists t_1\ t_0 \preceq t_1 \prec t \wedge \texttt{occ(}t_1\texttt{, leave(x))}|$$

$$\texttt{occ(}t_0\texttt{, arrive(Sally)))} = \texttt{exp(}t_0\texttt{, }t\texttt{, }\lambda\texttt{)}$$

where `norm(`μ`,`σ`,`u`,`v`)` gives the normal distribution $\mathcal{N}(\mu, \sigma)$ for each interval $[u, v]$.

Why did we not write `P(occ(`u`, arrive(Sally)))` instead of the less precise `P(occ(`u`, `v`, arrive(Sally)))`? The reason is simple. For a continuous-time domain, the *probability* of an event *at* a time point is always zero. We can only consider the *probability* that an event occurred in some interval. Of course, we *can* speak about the probability density function for the occurrence of an event at a point. The language **LL2** that we define below includes that feature. The reason that we do not do so for **LL1** is to simplify the definition of its semantics.

To obtain an estimate for the probability that Sally is still at the warehouse, regardless of what time she arrived there, we need the convolution as before. Note that this operational definition of the probability should be is not really expressible in **LL1**, unless we regard integration as a field function.

$$\texttt{P(holds(}t\texttt{, here(Sally))) =}$$
$$\int_{-\infty}^{t} \texttt{neg-exp(}z\texttt{, }t\texttt{, }\lambda\texttt{)} \times \texttt{norm(2pm, 15min, }z\texttt{)}dz$$

A key improvement of **LL1** over our previous theories is the ability to reason directly about order of events, and in general, about complex temporal relations. For instance, the probability that package `X` arrives before a deadline represented by the time constant `1pm` is:

$$\texttt{P(occ(U, arrive(package-X))} \wedge \texttt{U} \leq \texttt{1pm)}$$

We can also reason about things like the probability that the ranges of two facts overlap, and in general, an extensive set of temporal interval relations such as defined by Allen [1983]. Allen defines 13 interval relations, based on before, equal, meets,

overlaps, during, starts, and finishes, and their inverses (the inverse of equal is equal, and thus there are 13 instead of 14 relations). The interval relations are defined in terms of how the beginning and end points of one interval relates to the beginning and end points of another interval, where the relations are <, =, and >. Thus these relations are trivially represented in **LL1**. As an example, consider the relation `before`. `before`$(\pi,\ \xi)$ is true for two facts π and ξ iff π becomes true before ξ. *i.e.*,

$$\begin{aligned} \texttt{before}(\pi,\xi) &\equiv \texttt{occ}(u,\ \texttt{beg}(\pi)) \wedge \texttt{occ}(v,\ \texttt{beg}(\xi)) \wedge u \prec v \\ &\equiv \Delta(\texttt{beg}(\pi)) \prec \Delta(\texttt{beg}(\xi)) \end{aligned}$$

The probability of that expression is given by

$$\texttt{P}(\texttt{before}(\pi,\xi)) = \int_{-\infty}^{t} \texttt{P}(\texttt{occ}(t,\texttt{beg}(\pi))) \left[1 - \int_{z}^{t} \texttt{P}(\texttt{occ}(x,\ \texttt{beg}(\xi)))dx\right] dz$$

Note that we have not claimed that we can actually *compute* or even define that probability through the axioms of **LL1** itself. However, given knowledge about the facts π and ξ, for instance, it *is* easy to compute it outside the logic. But that is the province of the next chapter.

3.2.1 Semantics

The semantics for **LL1** is based on *possible worlds*. More precisely, it is based on *chronicles*, which are the analogue of possible worlds in a temporal context (see Section 2.2.3). The semantics for probability is given by a *probability measure* over the chronicles. The probability of an **LL1** fact is given by the sum of the probability measure of the chronicles in which that fact is true.

As with the temporal logics of McDermott and Shoham, we take the *meaning* of an event token as the (unique) *date* at which it occurs, and the *meaning* of a fact token as the (single maximal) interval over which it holds true. More precisely, the meaning of an event is a set of time point - chronicle pairs, and the meaning of a fact is a set of time interval - chronicle pairs. The meaning of an event token ε includes a time point - chronicle pair $\langle u, c\rangle$ iff ε occurs at time u in chronicle c. Similarly, the meaning of a fact token π includes a time interval - chronicle pair $\langle\langle u, v\rangle, c\rangle$ iff π is true between exactly time u and time v in chronicle c.

This set theoretic interpretation is made more explicit in the following notation. For a fact π, if its range is $[u, v]$ in chronicle c, then we write $\langle\langle u, v\rangle, c\rangle \in \pi$, and for an event ε, if its date is u in chronicle c, then we write $\langle u, c\rangle \in \varepsilon$. To simplify exposition, we often omit mentioning the chronicle. Thus, if a fact π has range $[u, v]$, then we write $[u, v] \in \pi$, and if an event has the date u, then we write $u \in \varepsilon$.

In the semantics below, we make reference to a measure function μ. μ is a non-negative function over the set of chronicles C. μ is a function such that $0 \leq \mu(c) \leq 1$ for all $c \in C$, $\mu(\emptyset) = 0$, and $\mu(C) = 1$. Thus $\mu : C \mapsto [0, 1]$. The way in which we give meaning to sentences involving probability is by the measure μ of the set of chronicles in which the sentence holds true. For instance, the probability of `holds(12pm, 1pm, lunch)` is the sum of the measure of all the chronicles in which lunch took place (at least) between 12pm and 1pm.

The semantics for event occurrence is similar, but there is a restriction on μ for events. An event only occurs at a unique point in each chronicle. To ensure this, the following must be true. Let $\mu([u, v] \in e)$ stand for the measure of all chronicles in which e occurs between $[u, v]$. Then for any $u < v < w$, $\mu([u, w] \in e) = \mu([u, v] \in e) + \mu([v, w] \in e)$.

The equivalent restriction does not hold for facts. This is because of the difference in the meaning of `occ` and `holds` that we noted earlier. Suppose $\mu([u, v] \in f)$ stands for the measure of all chronicles in which the fact f holds in $[u, v]$. Then for any $u < v < w$, it is *not* the case that $\mu([u, w] \in f) = \mu([u, v] \in f) + \mu([v, w] \in f)$. In fact, the sum of $\mu(I \in f)$ for two different disjoint intervals I can be greater than 1. For instance, suppose that f is true from noon to 2pm. Then $\mu([noon, 1pm] \in f) = \mu([1pm, 2pm] \in f) = \mu([noon, 2pm] \in f) = 1.0$.

We now present the semantics for **LL1**. We define an ***LL1*** *interpretation* to be a tuple $M = \langle C, O, R, T, F, D, \mu\rangle$, where

- C is the set of chronicles,
- O is the set of domain objects,
- R is a set of relations over the objects,
- T is a totally ordered set of time points,
- F is the set of field functions.
- D is a denotation function, and

- μ is the probability measure mapping $C \mapsto [0,1]$.

D is an aggregate function $D = \langle D_{\mathcal{O}}, D_{\mathcal{O}_p}, D_{\mathcal{P}}, D_{\mathcal{T}}, D_{\mathcal{F}}, D_{\mathcal{F}_f} \rangle$ where

- $D_{\mathcal{O}}$ maps every object in $\mathsf{O_C}$ to O,
- $D_{\mathcal{O}_p}$ maps every object predicate in $\mathsf{O_P}$ to R.
- $D_{\mathcal{T}}$ maps every element of $\mathsf{T_C}$ to T,
- $D_{\mathcal{F}}$ maps every element of $\mathsf{F_C}$ to $\mathbf{R}$, and
- $D_{\mathcal{F}_f}$ maps every element of $\mathsf{F_f}$ to F.

The meaning function D is a standard denotation function that associates with each (constant) symbol in the language **LL1** with the appropriate entity in the domain. As we stated earlier, we take the denotation of propositions to be the set of intervals and worlds in which that proposition holds. We associate a set of point - chronicle tuples $\langle u, c \rangle$ with each event and a set of interval - chronicle tuples $\langle \langle u, v \rangle, c \rangle$ with each fact.

This is done by defining each relation in R to be a mapping from a fixed set of objects to time - chronicle pairs. $R_{\mathcal{F}}$ is for facts, mapping to $2^{T \times T \times C}$, and $R_{\mathcal{E}}$ is for events, mapping to $2^{T \times C}$. We define the mapping $D_{\mathcal{P}}$ that maps from propositions to time - chronicle tuples as follows:

- $D_{\mathcal{P}}(\varphi(o_1, \ldots, o_n)) = D(\varphi)(D(o_1), \ldots, D(o_n))$

A similar meaning is given to field terms other than field constants. In other words, field terms composed of a field function $f \in \mathsf{F_f}$ applied to T-terms and F-terms. Each function in F is defined as an $n(i,j)$ mapping (we define an $n(i,j)$ mapping similarly as with the $n(i,j)$ function) $\bigcup_{i+j}(T^i \times F^j \to F)$. Before we assign precise semantics, we must be able to interpret variables in the time arguments to field functions.

A *valuation* V is an assignment of a value to each variable in the language. In **LL1**, this is simply a mapping $V : \mathsf{T_V} \to T$. An interpretation and a valuation together induce a meaning function DV. DV is a mapping such that

- $DV(\varphi) = V(\varphi)$ if $\varphi \in \mathsf{T_V}$,
- $DV(\varphi) = D_{\mathcal{P}}(\varphi)$ for $\varphi \in \mathcal{P}$,
- $DV(\varphi) = D_{\mathcal{F}\mathcal{F}}$ for field function terms,

- $DV(\varphi) = D_P$ for probability terms,
- $DV(\varphi) = D(p)$ otherwise.

$D_{\mathcal{FF}}$ is defined recursively as follows:

- $D_{\mathcal{FF}}(f(t_1, \ldots, t_n) = DV(f)(DV(t_1), \ldots, DV(t_n))$.

We are now finally ready to present the semantics for **LL1**, how we associate truth values with statements in the language. Let the expression φ^o/u mean a statement that agrees with φ everywhere except possibly at u. Given an interpretation M, a valuation v, and a chronicle c, $(M, v, c) \models \varphi$ as follows:

- $(M, v, c) \models t_1 \prec t_2$ iff $DV(t_1) \prec DV(t_2)$.
- $(M, v, c) \models \texttt{holds}(u, v, \pi)$ iff $\langle\langle u, v\rangle, c\rangle \in DV(\pi)$.
- $(M, v, c) \models \texttt{occ}(u, \varepsilon)$ iff $\langle u, c\rangle \in DV(\pi)$.
- $(M, v, c) \models \forall u\ \varphi$ iff $(M, v, c) \models \varphi^o/u$, for all possible u.
- $(M, v, c) \models \texttt{P}(\varphi) = f$ iff $DV(\texttt{P}(\varphi)) = DV(f)$.

We complete the definition by defining the field value of the probability function.

- $D_P(\texttt{P}(\varphi)) = \mu(\{c | c \in C, (M, v, c) \models \varphi\})$

An **LL1** interpretation M is a *model* for a wff φ, or $M \models_c \varphi$ if $(M, v, c) \models \varphi$ for all variable assignments v in chronicle c. A *sentence* is a wff without free variables. If a sentence φ is satisfied by an interpretation M under some variable assignment in chronicle c, then it is satisfied by M under *any* variable assignment in chronicle c. Thus, M is a model for φ. A wff φ is *satisfiable* if it has a model. A wff φ is *valid* ($\models \varphi$) if its negation is not satisfiable.

A note on the semantics. The above semantics for **LL1** is about as simple as it can be. First of all, the semantics for time and field entities is invariant given a variable assignment for time points. Given a fixed assignment to all time variables, the value of a time point and a time relation is the same in all possible worlds. This is also true for field constants and field functions. So if two time points $t_1 = t_2$ in one world, it is true in all worlds. Similarly, if `P(holds(t, `φ`)) = neg-exp2(t, `λ`)` for some assignment of t, it is true in all possible worlds under the same assignment.

Similarly, the definition of probability in **LL1** is invariant across all possible worlds. In all possible worlds, the probability of a wff in **LL1** is the same. Note therefore, that the interpretation of probability might be considered somewhat odd. In other words, in any number of possible worlds, it is possible that $(M, v, c) \models \varphi$. Let us say that the measure of all those possible worlds is 0.7, and thus, according to our semantics, `P(`φ`) = 0.7`. However, in those possible worlds in which φ is true, we would like to think that the probability of φ is 1.0. This discrepancy is easily fixed by extensions to **LL1** as we shall see later.

3.2.2 What it means

So now that we have defined **LL1**, giving its syntax and semantics, what does it all mean? What good is it for?

First of all, we have formalized the basic ideas that were present in `ODDS`, but without its restrictive independence assumptions. In the `ODDS` project, the language used to write down basic facts and causal rules was never fully specified. Nor did we provide the language with any formal semantics.

In **LL1**, we can express all that was possible to express in `ODDS`, and more. **LL1** is expressive enough in practice to use for reasoning about many problems. This particularly applies to domains where the problem is fixed, or where a particular problem has been instantiated, such that a propositional framework poses no handicaps. For instance, in **LL1**, it is easy to reason about orders of events, which was not directly addressed in `ODDS`.

Furthermore, **LL1** has a simple, easy to understand semantics. The way in which causal rules are written is somewhat different, because **LL1** has no time arithmetic or probability density functions. This is not grossly problematic, as both can be simulated by requiring the field functions specifying the probability of facts and events to do a little more work.

Second of all, with **LL1**, we have managed to come back full circle to an explicit temporal language, incorporating probability theory along the way. In **LL1**, we see how easy it is to incorporate the notion of chance and likelihood directly into a temporal language. We claim that this facilitates the evaluation of languages for use in commonsense temporal and causal reasoning.

Third of all, to jump ahead a bit, **LL1** gives us a solid base for extensions that enable us, for example, to reason about expected utilities. This is discussed in more

detail in the next section.

Finally, probability offers an alternate means of addressing the common problems in causal and temporal reasoning that we have mentioned before. **LL1** is adequate for describing and solving problems involving persistence and clipping of fluents, event and fact causation, and the qualification problem. Let us now explore this in detail.

We begin by considering the frame problem stated in probabilistic terms: "Does our model accurately capture our expectations regarding fluents that are considered *not* likely to change as a consequence of a particular event occurring?" The answer is yes insofar as frame axioms can be said to solve the frame problem in temporal logic; causal rules incorporating fact survivor functions are the probabilistic equivalent of frame axioms.

We note further that the **LL1** equivalent of frame axioms are more expressive. They give us a more exacting notion of how *likely* a fluent persists for various lengths of time.

In considering the ramification problem, "Does our model enable us to handle additional consequences that follow from a set of causal predictions?" For instance, if A is in box B and I move B to a new location, I should be able to predict that A will be in the new location along with B. **LL1** does not address this question directly. The basic idea of probabilistic inference can be extended to handle this sort of reasoning, but we have not investigated this to date.

The last problem we consider concerns reasoning about exceptions involving the rules governing cause-and-effect relationships. Does our model solve the qualification problem? That is to say, "Does our model accurately capture our expectations regarding the possible exceptions to knowledge about cause-and-effect relationships?" The answer is yes; in probability theory, an exception is just additional conditioning evidence that allows us to refine our predictions more accurately. It should be noted, however, that **LL1** imposes a considerable burden on the person setting up the causal rules. The **LL1** approach requires specifying all possible causes for each possible effect and the probability of each effect for every possible combination of possible causes. In general, it is not clear, however, that one can get away with less.

As an example of how this is handled, consider an example involving the coffee break as conditioning information. When we don't know about the coffee break, we predict Sally to leave relatively early. As soon as we learn about the coffee break, then our conditioning information changes, and thus our posterior expectation about

Sally's leaving time also changes. This is precisely the type of behavior that we would wish. While the only exceptions that can be handled are those that are already known, as we already suggested, it is far from clear that there is a better way.

In special instances, there may be *prototypical interactions* between facts and events that simplify the specification of causal rules. We have more to say about this when we discuss models of computation in Chapter 4.

3.2.3 Extensions

There are many ways in which we can extend **LL1**. Some extensions are simple syntactic extensions that greatly enhance expressiveness without necessitating major inventions in the semantics. Others involve more machinery for precise definition.

Enhanced fact/event ontology. For each fact π, add the enabling event `beg(`π`)` and clipping event `end(`π`)`. This considerably simplifies writing down axioms such as those for the truck driver. The semantics needs no augmentation as such for this extension. However, we need the following restriction. For each chronicle c and fact π,

$$\langle\langle u, v\rangle, c\rangle \in \pi \leftrightarrow \langle u, c\rangle \in \texttt{beg}(\pi) \wedge \langle v, c\rangle \in \texttt{end}(\pi)$$

This ensures that the semantics of enabling and clipping events match the intuitive definition.

More liberal field statements. Note that as it stands, the syntax for **LL1** forbids statements of the form `f1 = f2` for two different field terms `f1` and `f2`. It only permits assertions of the form `P(`φ`) = f`. An alternative, perhaps more natural definition, is to make `P(`φ`)` an F-term, and let `f1 = f2` be wffs. As a matter of fact, that is exactly what we do in **LL2**.

If we allowed the more liberal syntax with regard to field terms, then we can have arbitrary mathematical combinations of probability terms, like `P(holds(t + h,`φ`)) < 2 * P(holds(t + 2h,` φ`))`.

Again, such an extension does not require an augmented semantics as such. Instead of the rules defining satisfiability for `P(`φ`) = f` and we define essentially the same semantics for `f1 = f2`.

In **LL1**, we have tried to keep the syntax and semantics as simple as possible, while at the same time allowing a rich variety of causal theories. This is the only real reason that we have not incorporated the more expressive field syntax into **LL1**.

Measuring functions. This is related to the previous. A significant extension involves the definition of propositions by adding *measuring functions*. A measuring function, again, is a field function that describes quantities and other general numeric values other than probabilities. For instance, the height or weight of an individual.

An important measuring function is the *utility function* used in decision theory. Another important measuring function is the *date* function Δ that we have repeatedly mentioned. **LL**1 does not allow us to talk about the date of an event directly.

Once we have the date function, then we have yet another important measuring function, the *lifetime function* Λ that gives the duration of a fluent. Its definition is simple however, being just the difference of the dates of the enabling and clipping events of a fact:

$$\Lambda(\pi) = \Delta(\texttt{beg}(\pi)) - \Delta(\texttt{end}(\pi))$$

Separate measures μ for each possible world. We may adopt a *set* of measuring functions μ in the interpretation, that assign a different probability measure to each chronicle c. In such a logic, it is possible to speak of the probability of a statement can be different in different possible worlds, which fixes what might be considered an oddity in that we noted in the semantics of **LL1**.

Branching-time LL1. A further extension then is to define an accessibility relation over the possible worlds, so that we end up with a branching-time logic. This is akin to Haddawy's proposal [1990].

Time interval relation operators. We have already seen that **LL1** is capable of expressing complex temporal relations. It is possible to make those relations into modal operators in the spirit of Shoham [1988].

3.2.4 LLD

An interesting sublanguage of **LL1** results from restricting it to the representation of discrete intervals, rather than the whole continuous time line. We call the resulting language **LLD**. Whereas **LL1** is a language somewhat similar to `ODDS`, **LLD** is a language that is similar to the discrete model of probabilistic temporal reasoning that we presented in Section 2.3.2.

In **LLD**, the time line is partitioned into a set of discrete intervals I. Rather than time points, **LLD** sentences refer to intervals. We make the same type of distinction between fact and event as before. However, a key difference results from the restriction to intervals. We no longer have the ability to reason directly about fact holding or

event occurrence at a point. Thus, we never have statements of the form `occ(u, ε)`: we only have the interval variant `occ(u, v, ε)`. Similarly for facts. What this also means is that we lose the ability to make some distinctions. If two events occur in the same interval, their exact dates are *incomparable*. That is, we cannot distinguish accurately which occurred first. Thus, in practice, the partitioning of the time line that is used in a particular **LLD** domain theory should reflect the degree to which we *want* to be able to make such distinctions.

There are a number of variants within **LLD** that are interesting to consider. In general, we would allow causal rules to link facts and events to facts and events in any intervals. We may choose to restrict causal rules to link only facts and events in consecutive intervals. If we further only allow field constants for probability values, then the model becomes very similar to the one we presented in Section 2.3.2. Such models are very close to *Markov processes* as we show in Section 4.3.1.

We may also consider the same extensions to **LLD** that we considered for **LL1**. For instance, by expanding the scope of the field sort, we can add general real number functions, such as the decision theoretic utility function.

The semantics of **LLD** actually does not differ substantially from that of **LL1**. It is also based on possible worlds / chronicles, and is a straightforward exercise.

3.3 LL2

Now, we present a particular extension of **LL1** called **LL2**. In **LL2**, we widen the scope of the field and temporal sorts as we touched on in the previous section. For instance, we add time arithmetic as well as more liberal use of field functions and variables that we touched on in the previous section. More importantly, we incorporate probability density functions of time, and a new probability operator `f`. Finally, we incorporate enabling and clipping events of facts into the language from the beginning.

We mentioned previously that we do not as yet have a model theory for **LL2**. As with many so-called 'languages' in AI, all this means is that there is no *formal* semantics for the language. We *do* have a clear *intuitive* semantics, and probabilities for statements that we can express in **LL2** are easily computed. Although we would prefer to have a model theory, **LL2** is more of an exercise in creating a useful language for time information management. **LL1**, by contrast, was in large part an exercise in

formalizing ODDS, and in forging a link to past work in temporal logics.

In defining the syntax for **LL2**, again we start with a set of predicate symbols and function symbols of fixed arity. As before, we define below the entities in each of the three sorts.

1. The object sort:
   - $\mathsf{O_C}$: A set of object constants.
   - $\mathsf{O_P}$: Object predicates of various fixed arity.
2. The time sort:
   - $\mathsf{T_C}$: A set of time constants.
   - $\mathsf{T_V}$: A set of time variables.
   - $\mathsf{T_U}$: $\mathsf{T_C} \cup \mathsf{T_V}$.
   - $\mathsf{T_f}$: Time functions of various fixed arity.
   - $\prec$: The time precedence operator.
3. The field sort:
   - $\mathsf{F_C}$: A set of field constants.
   - $\mathsf{F_V}$: A set of field variables.
   - $\mathsf{F_U}$: $\mathsf{F_C} \cup \mathsf{F_V}$.
   - $\mathsf{F_f}$: Field functions of various fixed arity.
   - $<$: A field relation operator.

We also have the probability operators P and f. In addition, we have the measuring functions $\mathsf{F_M}$, which are functions of fixed arity with arguments of object sort, and which maps to the field sort. Finally, let $\mathcal{U} = \mathsf{T_U} \bigcup \mathsf{F_U}$. Given the above, we can define **LL2** inductively as follows.

The terms are:

- A single object constant or variable is an *O-term*. A single time variable or constant is a *T-term*. A single field constant or variable is an *F-term*.
- Let g be an $n(i,j)$-ary time function symbol. If $t_1, \ldots, t_i$ are T-terms, and $t_{i+1}, \ldots, t_n$ are F-terms, then $g(t_1, \ldots, t_n)$ is an T-term.

- Let g be an $n(i,j)$-ary field function symbol. If $t_1, \ldots, t_i$ are T-terms, and $t_{i+1}, \ldots, t_n$ are F-terms, then $g(t_1, \ldots, t_n)$ is an F-term.
- Let g be an n-ary measuring function symbol. If $t_1, \ldots, t_n$ are O-terms, then $g(t_1, \ldots, t_n)$ is an F-term.
- If φ is a wff not containing P or f, then P(φ) is an F-term.
- If φ is a wff not containing P or f, then f(φ) is an F-term.

We extend the definition of proposition for **LL2**.

DEFINITION 3.2
Let p be an n-ary object predicate symbol, and let $o_1, \ldots, o_n$ be O-terms. Then $p(o_1, \ldots, o_n)$ is a simple proposition.

We assume that a distinction is made between facts and events within the simple propositions, where $\mathcal{P}$ consists of disjoint sets, the facts $\mathcal{F}$, and the events $\mathcal{E}^0$.

DEFINITION 3.3
If $f \in \mathcal{F}$, then beg(f) and end(f) are derived events $\mathcal{E}^D$. The set of all events $\mathcal{E} = \mathcal{E}^0 \cup \mathcal{E}^D$. The set of all propositions $\mathcal{P}$is given by $\mathcal{E} \cup \mathcal{F}$.

Given this, the well-formed formulas (wffs) are defined as:

- If u and v are T-terms, then and $u \prec v$ are wffs.
- If u and v are T-terms, and $\pi \in \mathcal{F}$, then holds(u, v, π) is a wff.
- If u is a T-term, and $\varepsilon \in \mathcal{E}$, then occ($u, \varepsilon$) is a wff.
- If f_1 and f_2 are F-terms, then $f_1 < f_2$ is a wff.
- If φ_1 and φ_2 are wffs, then so are $\varphi_1 \wedge \varphi_2$, and $\neg\varphi_1$.
- If φ is a wff and $u \in \mathcal{U}$, then $\forall u\varphi$ is a wff.

As with **LL1**, we define the usual extended set of quantifiers and logical relations, as well as an expanded class of temporal and field relations. In addition, the *conditional density* is defined as

$$f(\xi|\varphi) = f(\xi \wedge \varphi)/f(\varphi)$$

LL2 allows the expression of substantially richer domain theories. First of all, we fix one defect of **LL1**, which was its lack of real probability density functions. Thus, the previous example we gave as 3.1 can be rewritten as follows.

$$\forall u \; \texttt{f(occ(}u\texttt{, arrive(Sally)))} = \texttt{normal(2pm, 15min, } u\texttt{)}$$

where `normal(`μ`,`σ`,`u`)` gives the normal distribution $\mathcal{N}(\mu, \sigma)$ for each point u.

The probability that a cold lasts longer than 3 days is written

$$\texttt{P}(\Lambda\texttt{(cold) > 3day}).$$

The utility of electing Dan Quayle President would be

$$\texttt{U(elect-president(Quayle))}$$

assuming that `U` is the *utility* function [von Neumann and Morgenstern, 1947]. The probability that the water level is greater than 3 meters at noon is

$$\texttt{P(holds(noon, water-level(x))} \wedge \texttt{ x > 3m}).$$

3.3.1 Extensions

The most interesting extensions to **LL2** are in features that facilitate planning and decision making. One important extension is the *expectation* operator `E`. As an example, such an extension allows us to talk about the expected duration of a cold `E(`Λ`(cold))`, or the expected utility of electing a Dan Quayle President

$$\texttt{E(U(elect-president(Quayle}))).$$

Other extensions include the ones in Section 3.2.3, such as branching worlds, and different measures for different possible worlds.

3.4 Related Work

We have already noted the relationship of our work with that of logics of probability, such as that of Bacchus, Halpern, Fagin, and others. Those logics have no special temporal component. Within AI, work in logics for reasoning about time and probability is a recent development.

3.4.1 Weber

Weber [1989a; 1989b] has developed a conceptually simple and clean framework for causal reasoning with uncertainty, based on the idea of *reference classes* [Kyburg, 1983]. Reference classes feature, essentially, a type of inheritance reasoning. A given proposition P may belong to a number of reference classes, some more specific than others. More specific classes yield more specific statistics for members of a given reference class. Weber's focus is on handling classical problems of temporal reasoning by utilizing such statistics. His *statistical causal rules* are a generalization of logical and probabilistic knowledge, including persistence and projection rules. Weber has developed a computation scheme called *highest impact first* which utilizes reference classes, by successively bounding a probability of interest starting with the most general reference class.

Weber gave the first account of probabilistic causal reasoning in AI based on logic. He did not provide a detailed syntax or semantics, instead embedding his new constructs within first-order logic. With Martin, mentioned below, and Bacchus, mentioned previous, Weber shares a statistical frequentist view of probability.

3.4.2 Haddawy

Haddawy [1990; 1991] has developed the *logic of time, chance, and action*, a logic for reasoning about plans. Haddawy's logic is substantially similar to van Fraassen's logic of objective chance [van Fraassen, 1980]. Within AI, Haddawy's logic is closest technically to our work in addition to being contemporaneous . Some of the basic ontology, syntax, and semantic features of Haddawy's logic and **LL** are similar. The logic of time and chance is a continuous-time logic with set-theoretic possible worlds semantics, much like **LL1**. They have both been developed with a view to planning applications. However, beyond the superficial similarities, the logics are different in important respects.

Much of the differences arise from the different goals for the languages. Haddawy is concerned with developing a logic for action, including distinct notions of action attempts and occurrences, and representing the interaction and feasibility of actions. By contrast, our development of the time net and the logic of lifetimes came at the same time, and we have developed the **LL** languages from the start with the clear aim of linking with Bayesian networks for efficient computation. We are concerned with developing a knowledge representation for time nets and for persistence and change.

Haddawy presents no computational theory for making inferences about facts and events over time. Although in his later work, Haddawy presents an algorithm for computing the earliest time of occurrence of events based on a theory, this is a restricted algorithm. Neither in his logic nor in his algorithm does Haddawy address the issue of how to reason about the changing probability of a fact or event over time.

There are many specific differences. Haddawy's notion of fact and event correspond to the traditional account in AI as distinct from the special notion of event that we have adopted for representing probability density functions. In Haddawy's logic, it only makes sense to talk about the probability of a fact or event relative to some particular point. To talk about the probability of some sentence φ, it is always in reference to a point t: $P_t(\varphi)$. Haddawy allows arbitrary nesting of such probability operators in his logic.

Haddawy's logic is restricted to reasoning about probability constants. Since he has no real number field functions, his framework is unable to express probability density functions of time. For a particular set of parameters and a particular discretization of time points, his logic would be capable of expressing, point for point, similar information that would be expressed in the logic of lifetimes, but such an approach would be exceedingly cumbersome.

3.4.3 van Fraassen

There have been logics for reasoning about time and probability in philosophy and in theoretical computer science predating the work in artificial intelligence. By and large, this work does not address issues of computation or of reasoning about continuous persistence.

van Fraassen [1980] presents a logic of *objective chance*. van Fraassen's notion of objective chance is that an expression about the probability of a fact is always *relative* to a particular point in time. Furthermore, objectively, a fact in the past relative to a point has either been true or false, so they can only have probability 0 or 1. The future, however, relative to a point in time *branches* according to how different possible worlds are accessible from others. Much of the ideas in Haddawy's logic are based on these ideas.

3.4.4 Probabilistic Dynamic Logic

In the theory of computer science, there have been attempts to develop temporal logics for reasoning about programs, circuits, and distributed systems. The most well-known of these is *dynamic logic*. Feldman [1984] has extended dynamic logic to include probabilities in his work on *probabilistic dynamic logic* (PDL). PDL is a propositional temporal logic with a semantics based on possible worlds. It does not have the notion of fact, event, or persistence that are useful in an AI action representation.

3.4.5 Martin and Allen

Martin and Allen [1991] have recently designed a language for planning using statistics of the success of actions in different situations. Their planning system maintains statistics about the result of actions in the form of confidence intervals. The confidence intervals are in terms of approximating binomial distributions by normal densities. Although their language notates actions and events relative to temporal intervals as in Allen's previous work [Allen, 1984], the actual temporal aspects do not appear to play a large role. In examples that they provide, there are statistics linking actions and effects, but they appear to hold for all time points. In other words, there is no difference in the probability of the effect given either the time of the action or of the effect, so time is basically irrelevant to their examples. They have not defined a semantics for their language.

Chapter 4

The Time Net

In this chapter, we explore a second kind of language for describing and reasoning about the world, the language of *time nets*. A time net is a *probabilistic network*, a (mathematical) graph for expressing the relationship between facts and the probability of facts. In a probabilistic network, each vertex represents a random variable, and sets of edges encode constraints between the random variables represented by the vertices. For example, suppose that you are trying to estimate the arrival time of your spouse's airplane flight from Los Angeles. In a time net for reasoning about this, one vertex may represent the time the flight left Los Angeles airport, and another vertex may represent the time of arrival. An edge between the two vertices would indicate that departure time affects arrival time (see Figure 4.1). A time net is used, first, to model a situation of interest in such a manner. Typically then, the probability distributions of some or all of the random variables in the graph are estimated with some algorithm. This process of *evaluating* a time net would, for instance, compute the probability distribution for each vertex, in our case, the time of departure and the time of arrival.

The graph abstraction of probabilistic networks separates qualitative issues of dependence between random variables from the purely quantitative aspects of the probability distributions of the random variables. Inquiry into the formal properties of probabilistic networks has found them quite robust [Lauritzen *et al.*, 1984; Pearl, 1988]. Clear relationships between graphs and probability distributions have been proven, making it easy to ensure that their use is consistent. On the practical side, by abstracting the *structure* of a domain visually from the quantitative aspects, we find that probabilistic networks make the tasks of construction and elicitation of a model of a domain easier. Another important computational benefit is the fact that the

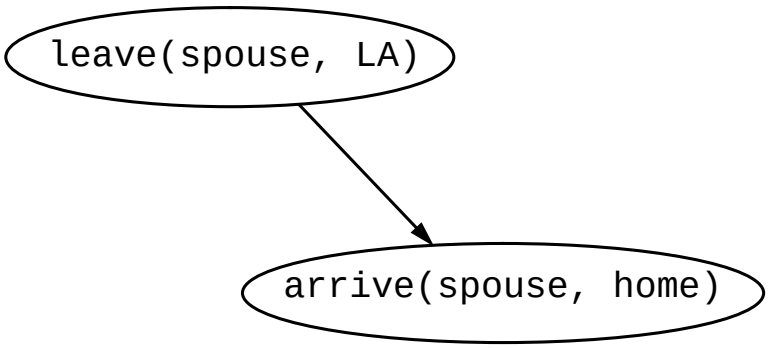


Figure 4.1: An example Time Net

graph is a compact way of specifying and storing the probability distributions. Last but not least, graph representations are neutral as far as interpretation of probability, allowing research to focus on modeling and problem solving.

Time nets belong to a popular directed acyclic graph class of probabilistic networks called *Bayesian networks*. There are also undirected probabilistic networks, called Markov random fields, that have been used for temporal inference [Dean and Kanazawa, 1988b]. However, we focus on Bayesian networks exclusively in this dissertation. In the remainder of this chapter, we explore the expressive and computational properties of time nets. First, we introduce graph concepts and the Bayesian network. Then, we present the time net itself. Finally, we discuss algorithms for evaluating time nets.

4.1 Graph Notation and Concepts

First, we present some graph notation. This presentation follows that of Tarjan [Tarjan and Yannakakis, 1984] and Lauritzen [Lauritzen *et al.*, 1984].

A *graph* is a tuple $G = (V, E)$, composed of a finite set of *vertices* V, $n = |V|$, and a finite set of *edges* $E = \{(v, w) | v \in V, w \in V, v \neq w\}$, $e = |E|$. Thus $E \in 2^{V \times V}$. We use *node* interchangeably with vertex and *arc* interchangeably with edge.

A graph is called a *directed graph* when every edge $e \in E$ is an ordered pair. In a directed graph, an edge is usually represented as an arrow. The direction that the arrow points is *down (the graph)*, and the direction pointing against the arrow is *up (the graph)*. A graph is an *undirected graph* if the edges are unordered pairs. In an undirected graph, if $(v, w) \in E$, then we may consider $(w, v) \in E$ to be true as well.

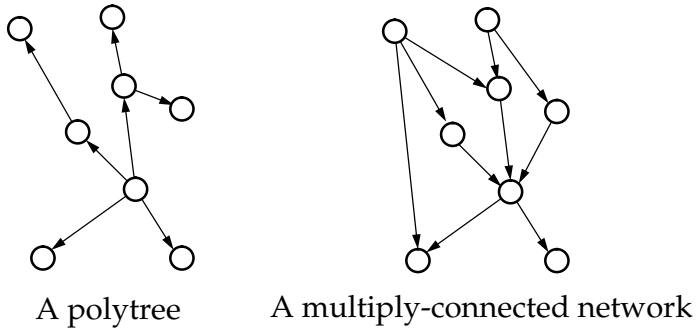


Figure 4.2: A polytree and a multiply-connected graph

In both directed and undirected graphs, if $(v, w) \in E$, then v is a *neighbor* of w. The neighbor relation is reflexive, *i.e.*, if v is a neighbor of w, then w is a neighbor of v. Whenever v is neighbor of w, v is *adjacent* to w. The set of all neighbors of w is $Adj(w)$.

In a directed graph, if $(v, w) \in E$, then v is a *parent* of w, and w a *child* of v. The set of all parents of a vertex v is denoted $Pa(v)$ and the set of all children $Ch(v)$. If $Pa(v) = \emptyset$, then v is a *root* node. If $Ch(v) = \emptyset$, then v is a *leaf* node.

A *path* of length k from v_0 to v_k is a sequence of vertices $\{v_0, v_1, \ldots, v_{k-1}\}$ such that $(v_{i-1}, v_i) \in E$ for $i = 1, \ldots, k$. In a directed graph, if there is a path from v to w, then v is an *ancestor* of w, and w is a *descendant* of v. The set of all ancestors of w is denoted $An(v)$, and the set of all descendants of v is denoted $De(v)$.

If a directed graph contains a path from v to v for some $v \in V$, then the path is a *cycle*. Otherwise, the graph is *acyclic* and it is called a *directed acyclic graph* or *dag* for short.

A graph with only one path connecting any two nodes is called a *polytree*, or *generalized Chow tree* [Chow and Liu, 1968]. We say that a graph is *singly-connected* when it is a polytree. Otherwise, it is *multiply-connected* (see Figure 4.2).

An undirected graph obtained from a directed graph G by ignoring the ordering on the pairs of vertices in E is the *undirected graph corresponding to* G. If $G' = (V, E')$ is the undirected graph corresponding to $G = (V, E)$, then if $(v, w) \in E$, then $(v, w) \in E'$ and $(w, v) \in E'$.

When a graph is multiply-connected, there is more than one path connecting some two nodes in a graph. If the graph G is a dag, then the undirected graph

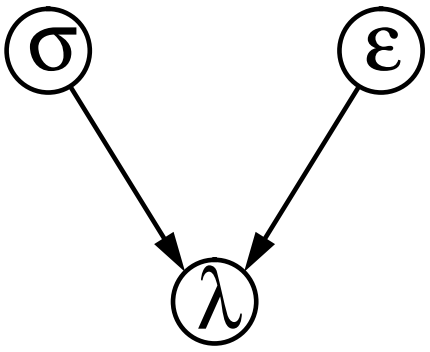


Figure 4.3: A Bayesian Network G

corresponding to G contains a *loop*, or *undirected cycle*.

If $V' \subseteq V$, then V' induces a subgraph $G_{V'} = (V', E_{V'})$ where $E_{V'}$ is that subset of E restricted to $V' \times V'$. An undirected graph (V, E) is *complete* iff $\forall v, w \in V (v, w) \in E$. If $V' \subseteq V$ induces a complete subgraph, then V' is said to be complete. A maximal complete graph with respect to set inclusion is a *clique*.

For the graph (V, E), we define its *moral graph* as the undirected graph with the same vertex set in which v is adjacent to w just in case either $(v, w) \in E$, $(w, v) \in E$, or there exists $x \in V$ such that both $(v, x) \in E$ and $(w, x) \in E$.

A subset $S \subseteq V$ is said to *separate* $X \subseteq V$ from $Y \subseteq V$ if every path from a vertex in X to a vertex in Y intersects S.

A *hypergraph* $H = (V, E)$ consists of a set of *vertices* V and a set of *edges* E. Each edge $e \in E$ is a subset of V. A graph is a hypergraph all of whose edges have size two. To distinguish between a hypergraph and a graph, we may call an edge for a hypergraph a *hyperedge*.

4.2 Bayesian Networks

A *Bayesian network* (*Bayesian belief network*) is an augmented dag for representing a set of random variables and their probability distributions. Directed graph models of probability similar to Bayesian networks were apparently first formally investigated by Wright in his work on *path analysis* in genetics ([Wright, 1921] as cited in, *e.g.*, [Lauritzen *et al.*, 1984]). An excellent introduction to Bayesian networks, and to probabilistic networks in general, is Pearl's book [1988]. Another excellent text on Bayesian networks is Neapolitan's book [1990].

Informally, a Bayesian network is a dag $G = (V, E)$. Each $v \in V$ corresponds to a random variable that we are interested in modeling. Where unambiguous, we refer to a vertex and the random variable that the vertex corresponds to by the same name. Each edge $(v, w) \in E$ indicates *direct dependence* of v on w. What this means exactly is seen later. Figure 4.3 shows an example of a simple Bayesian network. This Bayesian network models the relationship of lung cancer to smoking and nutrition. Without loss of generality, we assume that all Bayesian networks are *connected*, as in this example.

The Bayesian network G contains three nodes $V = \{\sigma, \epsilon, \lambda\}$, representing three propositional (true/false) random variables indicating whether or not a person smokes (σ), whether or not she eats badly (ϵ), and whether or not she has lung cancer (λ). Random variables in a Bayesian network do not have to be binary propositions. For example, the different kind of eating habits might be `vegan`, `ovo-lacto`, `fish-and-chicken`, and `red-meat`, instead of just `eat-badly` and ¬`eat-badly`. In fact, as we shall see, a Bayesian network random variable may take on its value from a real-valued range (a *continuous random variable*), instead of a finite set (a *discrete random variable*). For instance, a random variable may be the number of calories an individual consumes per day.

G contains two edges $E = \{(\sigma, \lambda), (\epsilon, \lambda)\}$, indicating that the probability of lung cancer depends directly on smoking and on nutrition. The relationship between the random variables is: (1) smoking increases the chances of lung cancer, (2) bad nutrition in the presence of smoking increases it further, and finally, (3) without smoking, nutrition slightly increases the chances of lung cancer.

This knowledge is stored in the form of probability matrices $\mathcal{P}$ at each node $v \in V$. The probability matrices at the root nodes σ and ϵ store, respectively, the prior unconditional probability that a random person smokes, and the prior unconditional probability that a random person eats badly. These distributions are commonly called *marginal distributions*. The probability matrix at the non-root node λ is a *conditional distribution*, giving the probability of lung cancer depending on the values of its parent nodes. Tables of the probability matrices appear in Table 4.1.

A point about Bayesian networks often found confusing is that there is only *one* conditional probability table for a non-root node. Naively, it seems plausible that there should be a table for each edge going into each non-root node. For instance, in our example, it may seem that what we need is to specify the probability of lung

Probability of σ	
Smokes	$P(\sigma)$
yes	0.3
no	0.7

Probability of ϵ	
Eats badly	$P(\epsilon)$
yes	0.6
no	0.4

Probability of λ given σ and ϵ			
Smokes	Eats badly	Has lung cancer	$P(\lambda\|\sigma,\epsilon)$
yes	*yes*	*yes*	0.9
yes	*yes*	*no*	0.1
yes	*no*	*yes*	0.7
yes	*no*	*no*	0.3
no	*yes*	*yes*	0.15
no	*yes*	*no*	0.85
no	*no*	*yes*	0.1
no	*no*	*no*	0.9

Table 4.1: The probability matrix tables for G

cancer given smoking taken alone, and then also to specify the probability of lung cancer given eating habit alone. However, this is not case. The Bayesian network formalism requires that there is one table, specifying the probability of lung cancer given smoking and eating habit taken together.

What this boils down to is that it is often better to regard a Bayesian network as a hypergraph, rather than as a graph. Each hyperedge in the hypergraph is a set of edges all "meeting" at the same vertex. For each hyperedge in a Bayesian network, there is a conditional probability table. This also defines what we mean by direct dependence: that the random variable corresponding to a vertex that a hyperedge points to is directly dependent on all of the random variables corresponding to the vertices that the hyperedge points from.

The graph G along with the probability matrices at each node formally define a Bayesian network B. B defines a unique, consistent *joint distribution* P over the random variables V. B is a particular *factorization* of the distribution P. What we mean by this, illustrating with our example Bayesian network, is that the joint distribution function $P(\sigma, \epsilon, \lambda)$ over all the random variables is uniquely determined by the three probability matrix tables $P(\sigma)$, $P(\epsilon)$, and $P(\lambda|\sigma, \epsilon)$. In particular, the relationship between them is given by the following product:

$$P(\sigma, \epsilon, \lambda) = P(\lambda|\sigma, \epsilon)P(\sigma)P(\epsilon).$$

We now develop some of these concepts more formally.

Let R be a finite set of n random variables $\{r_0, r_1, \ldots, r_n\}$. Each $r \in R$ has a *state space* Ω_r, a mutually exclusive and exhaustive set of values that r can take on. r takes on exactly one value out of Ω_r at any one time. Ω_r can be either continuous or discrete. A continuous random variable has a state space, or *range*, that is a possibly infinite subset of the real line:

$$\Omega_r \in (-\infty, +\infty)$$

Each discrete random variable has a finite set of m_r values:

$$\Omega_r = \{\omega_0, \omega_1, \ldots, \omega_{m_r-1}\}$$

We will call m_r the *arity* of the random variable r; a random variable of arity n is referred to as a *n-ary random variable*. The set Ω_r will sometimes be referred to as the *alternate set* of r, and each member of Ω_r an *alternate* of r.

The notions of state space, alternate set, and alternate are extended to sets of random variables $r^{(k)} = \{r_0, \ldots, r_{k-1}\}$. The state space of such a set is $\Omega_{r^{(k)}}$

$$\Omega_{r^{(k)}} = \omega_{r_0} \times \ldots \times \omega_{r_{k-1}},$$

and an alternate is a particular set of values for the r_i. A special case is the state space of the whole set of random variables R, called the *configuration space* Ω:

$$\Omega = \omega_{r_0} \times \omega_{r_1} \times \ldots \times \omega_{r_n}$$

A *probability distribution* for a discrete random variable $r \in R$ is a function that assigns a probability to each state of r

$$P(r = \omega),$$

$\omega \in \Omega_r$ such that the value is non-negative,

$$\forall \omega \in \Omega_r \; P(r = \omega) \geq 0$$

and the sum over the state space is 1:

$$\sum_{\omega \in \Omega_r} P(r = \omega) = 1$$

The *joint probability distribution* for a set of k random variables $\{r_0, \ldots, r_{k-1}\} \subseteq R$ (for discrete r_i) is a probability distribution over the alternates in the joint state space $\Omega_k = r_0 \times \ldots \times r_{k-1}$:

$$P(r_0 = \omega_0, \ldots, r_{k-1} = \omega_{k-1})$$

Finally, the *conditional probability distribution* of a random variable r, dependent on a set of k *antecedents* $c = \{c_0, \ldots, c_{k-1}\} \subseteq R - r$ is

$$P(r = \omega | c_0 = \omega_0, c_1 = \omega_1, \ldots, c_{k-1} = \omega_{k-1})$$

for each alternate in the alternate sets for r and c.

For simplicity, we normally express the probability that a random variable takes on a particular value ω in its state space as

$$P(\omega)$$

and the function in general as

$$P(r).$$

We adopt similar shorthand for the joint probability and conditional probability as well, letting a sequence of variables or values be denoted by single letters. For example, let z be a vector of random variables $\{z_0, \ldots, z_k\}$. Where unambiguous, we allow

$$P(z)$$

as shorthand for the expression

$$P(z_0 = \omega_0, \ldots, z_k = \omega_k)$$

We are now ready to provide our first formal definition of a Bayesian network.

DEFINITION 4.1
A Bayesian network is a tuple $B = (V, E, \mathcal{P})$, comprised of the dag $G = (V, E)$, and a set of probability functions $\mathcal{P}$. V is a set of random variables. $\mathcal{P}$ is a set of n functions, one for each $v \in V$, mapping from the joint state space of conditioning set $c_v = \{w|(w, v) \in E\}$ and v to the interval $(0, 1)$. The probability function $p_v \in \mathcal{P}$ for v gives the probability of each alternate of v for each alternate in the joint state space of c_v:

$$P(v|c_v)$$

As noted earlier, the Bayesian network B is a factorization of the joint probability P over V. In general, a joint probability distribution can be factored in the following form:

$$P(v_0, v_1, \ldots, v_{n-1}) = P(v_0|\ldots)\ldots$$

In a Bayesian network, this is simplified to this *product rule*:

$$P(v_0, v_1, \ldots, v_{n-1}) = \prod_i P(v_i|c_{v_i}).$$

We will not go into the proof of this here.

Because direct dependence is always indicated by an arc, Bayesian networks make it easy to identify the conditional independence in a model. Indeed, Pearl has argued [1988] that the importance of the Bayesian network lies not in the numerical probabilities in the probability matrices, but in the qualitative dependence information provided in the *structure* of the graph. The Bayesian network is only one class of graphical models about probabilistic dependencies, called *dependency models*. Some dependency models, such as cyclic theories, or complex mutual dependencies cannot be represented in a Bayesian network. For more details on these issues, see [Pearl, 1988]

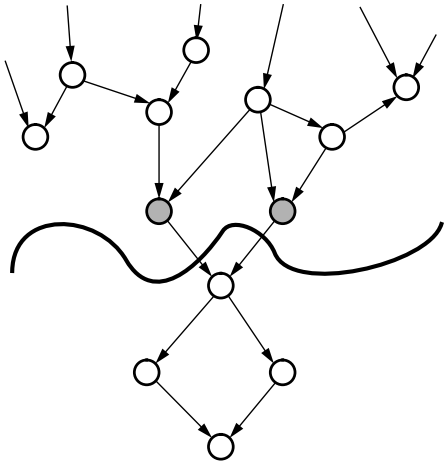


Figure 4.4: Independence in a large Bayesian network.

Identifying independence has important consequences for computation. Suppose that there is a Bayesian network consisting of hundreds of nodes, and we are interested in the probability of one of the random variables. If we can determine that this particular random variable is only dependent on a handful of the nodes, then it is unnecessary to consider the whole network for the purposes of computation. Such relationships can realize large computational savings.

As an example consider Figure 4.4. The two grey vertices (as opposed to all the others, which are white), are assumed to be nodes whose values are known. For instance, we know the blood cell count and body weight of an individual exactly. We call these the *evidence nodes*. The evidence nodes*separate* the network into two parts. The part 'above', which models all the factors and indicate what influences blood cell count and body weight, and the part 'below', which is influenced by blood cell count and body weight.

Ordinarily, the 'above' part influences the 'below' part through the evidence nodes. However, when the value of the evidence nodes are known, in this case, they "block" the influence pathways. As a result, in order to evaluate the small 'below' part of the network, we can ignore the 'above' part.

Pearl *et al.*. [1989] provide formal results pertaining to independence relationships provided by a Bayesian network. Their criterion, called *d-separation* makes it possible to determine when two random variables are independent in terms of graphical criteria on the Bayesian network for the random variables. As might be guessed from the above example, the d-separation criterion is closely related to graph separation.

As it happens, Lauritzen [1988] shows another criterion for determining independence that is equivalent to d-separation, and which directly uses graph separation.

4.3 The Time Net

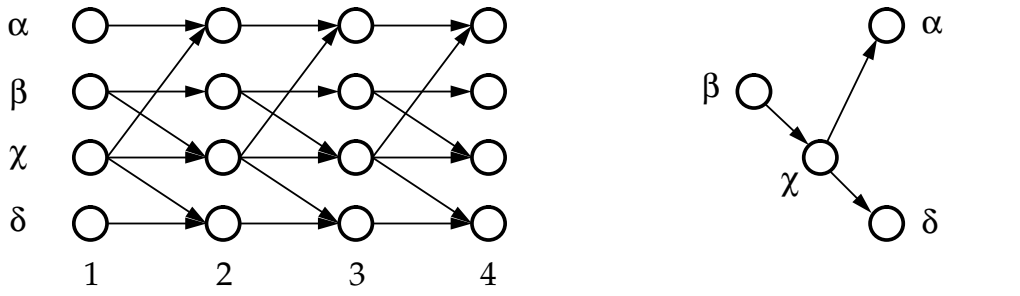
We adopt *time net* as the generic name for a graph used for reasoning about time. In this dissertation, we are concerned solely with the *probabilistic time net*, a variant of time nets used for reasoning about probability and time. The other kind of time net, the *deterministic time net*, is not explored here. Unless otherwise noted, a time net is a probabilistic time net in this dissertation.

The time net is a special class of Bayesian network developed for reasoning about time.[1] A time net captures knowledge about facts and events, their probability, and how they affect other facts and events. Any Bayesian network where the random variables predominantly make a reference to time is a time net. A random variable in a time net typically indicates whether or not a fact holds true or an event occurs at some point of time, or over some interval of time. As with any Bayesian network, we may enter evidence, and compute the distributions of the random variables.

A given time net may represent all knowledge relevant to a problem solving domain, or more typically, it may focus only on the knowledge relevant in a particular situation. For any problem solving domain, there are typically a number of time nets applicable for different contingencies arising in the domain. An interesting research area concerns the construction of time nets, and probabilistic networks in general, from a body of knowledge about a problem solving domain. Such knowledge may be expressed in a logic or similar language of the kind explored in Chapter 2. The issue of knowledge-based construction of time nets is explored in Chapter 5. This section focuses on the basic properties of the time net formalism.

There are two types of time nets, *discrete time nets* and *continuous time nets*. A pair of simple discrete time net and continuous time net is shown in Figure 4.3. The main differences between them are the model of time and the type of random variables involved. A discrete time net utilizes a discrete model of time, with a partition of time into discrete intervals. A node in a discrete time net typically represents the probability that a fact holds, or that an event occurs, during each of these intervals.

[1] In past work, time nets were called *temporal Bayes nets* [Dean and Kanazawa, 1989a].



(a) discrete time net (b) continuous time net

Figure 4.5: Time Nets

Each node for expressing such a probability can be discrete-valued or continuous-valued. By contrast, a continuous time net has a continuous model of time. A node in a continuous time net typically represents the time at which an event occurs, or the time at which a fact becomes true or false. A node representing the time of an event in a continuous time net must be continuous-valued. Roughly speaking, the continuous time net is more compact and expressive: it is easier to represent facts, events, and their durations and probabilities in a continuous time net than in a discrete time net. Furthermore, a continuous time net is better suited to capturing the kind of knowledge expressible in the more expressive logics presented in Chapter 2. The discrete time net was developed earlier historically, however, and we present it first.

The discrete time net was developed for computing the probability of facts and events such as with the discrete model of probabilistic temporal reasoning developed in section Section 2.3.2. The continuous time net was developed as a further evolution of the computational model of ODDS described in Section 2.3.1.

In the following discussion, let facts, events, and the enabling and clipping events of facts be defined as in Chapter 2. Conceptually, in our theory, time has infinite extent, stretching without bound both forward and backward. In practice, in computing the chance of events and facts over time, we generally restrict our attention to a finite subset of the time line. In such a case, there is assumed to be an *earliest time of interest* eti , and a *latest time of interest* lti. In such a case, inferences are limited to the interval $(\mathsf{eti}, \mathsf{lti})$, possibly augmented with the special time point ∞, which is taken to mean "never".

In a similar way, our knowledge may theoretically incorporate a model of an

infinite number of facts and events. Translating such a model into a time net can involve creating an infinite time net. In practice, we assume that a time net used for reasoning about a particular situation is finite and consist of a finite number of facts and events.

4.3.1 The Discrete Time Net

A discrete time net is a Bayesian network that uses discrete-valued random variables to model the states and change over time of some phenomenon of interest in the world. Each particular discrete time net is a model of the world. As noted earlier, a Bayesian network is generally used to reason about a particular instance of some problem domain. Our approach for defining a discrete time net is to do so in terms of the knowledge applicable in such an instance.

Let a *domain instance*, or *domain*, be a tuple $D = (\mathsf{eti}, \mathsf{lti}, \mathcal{T}, \mathcal{F}, \mathcal{E}, \mathcal{S}, \mathcal{P})$ consisting of a *time partition* $\mathcal{T}$, *fact tokens* $\mathcal{F}$, *event tokens* $\mathcal{E}$, *alternate sets* $\mathcal{S}$, and *probability functions* $\mathcal{P}$.

A time partition of size n is a sequence of time points $\mathsf{eti} = \tau_0 < \tau_1 < \tau_2 < \ldots < \tau_n = \mathsf{lti}$. The underlying ontology could support either a discrete or continuous interpretation of the time points. The *time intervals of* D, $\mathcal{I}_D$ (or simply the *time intervals*, $\mathcal{I}$), is the set of intervals $(\tau_{i-1}, \tau_i]$ for i from 1 to n. A *time interval* $\iota \in \mathcal{I}$ is one of those intervals, and *i-th time interval of* $\mathcal{I}$ ι_i is $(\tau_{i-1}, \tau_i]$. For $\iota_i \in \mathcal{I}$, its *predecessor* $Pred(\iota)$ is the ι_{i-1}, and its *successor* $Succ(\iota)$ is ι_{i+1}. The exceptions are ι_1, which does not have a predecessor, and ι_n, which does not have a successor.

$\mathcal{F}$ and $\mathcal{E}$ are facts and events of interest. As before, a fact is something that can hold over some duration of time, *i.e.*, over one or more intervals $\iota \in \mathcal{I}$. By contrast, an event can only occur during an infinitesimally small interval of time, *i.e.*, in at most one $\iota \in \mathcal{I}$. There are few restrictions on $\mathcal{F}$ and $\mathcal{E}$: either or both can be empty, and there is no general requirement, for example, that if a domain instance contains a fact, then it must also contain the fact's *enabling* and *clipping* events (cf. Chapter 2). It is also possible for $\mathcal{F}$ to contain facts involving continuous random variables. To simplify discussion, assume that all random variables are discrete for the moment.

There are two ways of representing each fact or event as a random variable. Let us talk about a fact φ for now. The case for an event is basically the same. Further, let us assume that the time interval of interest is `(u, v)`.

One possibility is to regard the status of whether or not φ holds as a random

variable. So there are two possibilities, `holds(u,v,`φ`)`, and $\neg$`holds(u,v,`φ`)`. The possibilities are mutually exclusive and exhaustive, and the sum of their probabilities sum to 1. Thus we can have a random variable of form $\pm$`holds(u,v,`φ`)`, where $\pm$ is meant to indicate that the random variable is a binary random variable that takes on values true and false.

This is all well and good when φ is by nature a true/false kind of fact. Suppose, though, that φ is a proposition of form $\phi(\{\zeta\})$, where $\{\zeta\}$ is a set of variables and constants with at least one variable, and where there are at least 3 different possible assignments to the variables in $\{\zeta\}$. For example let φ stand for "the color of my Miata", `Miata-color(`γ`)`, where γ can be one of {`white, blue, red, silver`}. In such a case, there is another natural way to form a random variable for φ. Rather than having four binary random variables $\pm$`holds(u,v,Miata-color(red))` and so on for each color, it would be natural to have a single 4-*ary* random variable `holds(u,v,Miata-color(?))`, where the `?` is meant to indicate that it may take on a number of alternates, in our case, four alternates.

For discrete time nets, we generally adopt this latter method of forming *n-ary* random variables from the underlying set of fact tokens and event tokens. Let a group of mutually exclusive and exhaustive alternatives such as in our example be an *aset*. Then the *alternate sets* or *asets* $\mathcal{S}$ for D is the union of all the asets for a domain instance. Each member of an aset is an *alternate*. Exactly one, and only one, of the alternates in an aset can be true at any given time. An aset contains either only fact tokens or only event tokens. Furthermore, each fact token and each event token must belong to one and only one aset. $\mathcal{S}$ comprises two subgroups formed from $\mathcal{F}$ and $\mathcal{E}$, $\mathcal{S}_F$ and $\mathcal{S}_E$. Usually, some descriptive name would be given to each aset, for instance "`Miata-color`" or something related for the previous example. Later on, when we link logic and time nets together, we show a syntactic basis for the formation of asets (and their names). The discrete time net formalism itself contains no notion of syntax for the random variables that it represents.

An aset defines the state space of a random variable without regard to a specific time interval. A *reified aset*, or *raset*, associates an aset with a time interval. Let the *asets at interval* ι $\mathcal{S}_\iota$ be the union of all rasets associated with interval ι. Let then the *domain asets*, $\mathcal{SS}$, be defined by $\mathcal{SS} = \bigcup_{\iota \in \mathcal{I}} \mathcal{S}_\iota$.

We associate with each $\sigma \in \mathcal{SS}$ a *probability function* π_σ. The set of all probability functions $\mathcal{P} = \bigcup_{\sigma \in \mathcal{SS}} \pi_\sigma$. Each π_σ is a function mapping from a (possibly empty)

conditioning set Γ_σ and an alternate $\xi \in \sigma$ to the real interval $(0, 1)$. Γ_σ is any set of asets $\gamma \in \mathcal{SS}$ such that $\gamma \neq \sigma$. A particular set of alternates, one for each $\gamma \in \Gamma_\sigma$, is a *conditioning instance* of σ. π_σ over $\xi \in \sigma$ must sum to 1 for each conditioning instance. The intended meaning of a distribution function is the following. For each alternate ξ for some $\sigma \in \mathcal{SS}$,

$$P(\sigma = \xi | \gamma_0 \wedge \ldots \wedge \gamma_m) = \pi_\sigma(\gamma_0, \ldots, \gamma_m, \xi)$$

where $\{\gamma_0, \ldots, \gamma_m\} = \Gamma_\sigma$. If Γ_σ is empty, then π_σ is a marginal distribution; otherwise, it is a conditional distribution. Note that we have placed no restriction on the intervals of the conditioning set with respect to the conditioned aset. This means that, in general, conditioning on information on "future" events is allowed. In practice, there may be more restrictive assumptions.

Given the above, we can now define the discrete time net.

DEFINITION 4.2

A discrete time net for D is a Bayesian belief network $B = (V, E)$. V consists of one node for each aset in $\mathcal{SS}$. E consists of a set of edges for each $\pi \in \mathcal{P}$ as follows. For each $\pi_\sigma \in \mathcal{P}$, add the edge (γ, σ) for each $\gamma \in \Gamma_\sigma$. E is the union of all such edges. As with asets, we may define the nodes for interval ι, V_ι, consisting of the nodes for $\S_\iota$.

That defines a discrete time net in a general form. There are two restrictions that must be present for a discrete time net to be *proper*. First of all, the set of edges E must be acyclic, so that the result is a proper Bayesian network. Secondly, if D contains any event tokens, then for each event token, the probability over all intervals of the event token must sum to less than or equal to 1. The latter ensures, among other things, that an event takes place during at most one interval.

Let us consider a simple example of a domain instance, and the corresponding discrete time net. For simplicity, we consider only four time points, producing a time net with three time intervals. The time points that we consider are on the hour every hour from noon through 3 p.m.. We let `0` stand for noon, `1` stand for 1 p.m., and so on.

$$\begin{aligned} \mathsf{eti} &= \mathtt{0} \\ \mathsf{lti} &= \mathtt{3} \\ \mathcal{T} &= \{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}\} \\ \mathcal{I} &= \{\mathtt{(0,\ 1]}, \mathtt{(1,\ 2]}, \mathtt{(2,\ 3]}\} \end{aligned}$$

We will number the time intervals as consecutive integers from 0. Thus, `(0, 1]` is interval 0, `(1, 2]` is interval 1, and so on.

Our example is a trip in a bus from New York to Atlantic City. There is only one fact token:

$$\mathcal{F} = \{\texttt{loc(Taj)}\},$$

which stands for the fact of our being in the Taj Mahal in Atlantic City. There are more event tokens. In addition to the enabling and clip events for `loc(Taj)`, there are the events of leaving New York City, and arriving at the Taj Mahal.

$$\mathcal{E} = \{\texttt{beg(loc(Taj))}, \texttt{end(loc(Taj))}, \texttt{leave(NYC)}, \texttt{arrive(Taj)}\}$$

We omit listing the asets $\mathcal{S}$ for this example in full as they are all binary propositional random variables. For instance, there is the aset for a node named `leave(NYC)` with value true and false. The domain asets consist of $\mathcal{S}$ $\times$ the time intervals. For a fact or event φ, we adopt the notation φ`[n]` to mean the *aset* named φ for time interval `n`. Then, the domain asets $\mathcal{SS}$ consists of:

`leave(NYC)[0]`	`leave(NYC)[1]`	`leave(NYC)[2]`
`arrive(Taj)[0]`	`arrive(Taj)[1]`	`arrive(Taj)[2]`
`beg(loc(Taj))[0]`	`beg(loc(Taj))[1]`	`beg(loc(Taj))[2]`
`end(loc(Taj))[0]`	`end(loc(Taj))[1]`	`end(loc(Taj))[2]`
`loc(Taj)[0]`	`loc(Taj)[1]`	`loc(Taj)[2]`

The pattern of dependence is relatively simple. We will not give the full set of probability functions for the asets, as this involves a total of 15 probability matrices. However, we do list here a table (Table 4.2) of the pattern of dependence between all the asets.

First of all, the time leaving New York City depends on nothing per se. It is given by a prior distribution at each interval. Any set of distributions is fine as long as the sum of the probability of leaving over all intervals is at most 1. Similarly for the end of being at Atlantic City. In this small example, once we arrive in Atlantic City, nothing causes us to leave. We stay there indefinitely.

Arriving at Atlantic City, however, depends on many factors. At interval 0, there is no chance of arriving (this is just how we set up the model). At interval 1, it depends on whether or not we left NYC in the previous interval, and also on whether we had already arrived in the previous interval. At interval 2, it depends on the same

Aset	Depends on
leave(NYC)[0]	$\emptyset$
leave(NYC)[1]	$\emptyset$
leave(NYC)[2]	$\emptyset$
arrive(Taj)[0]	$\emptyset$
arrive(Taj)[1]	leave(NYC)[0], arrive(Taj)[0]
arrive(Taj)[2]	leave()[0], leave()[1], arrive()[0], arrive()[1]
beg(loc(Taj))[0]	arrive(Taj)[0]
beg(loc(Taj))[1]	arrive(Taj)[1]
beg(loc(Taj))[2]	arrive(Taj)[2]
end(loc(Taj))[0]	$\emptyset$
end(loc(Taj))[1]	$\emptyset$
end(loc(Taj))[2]	$\emptyset$
loc(Taj)[0]	beg(loc(Taj))[0], end(loc(Taj))[0]
loc(Taj)[1]	loc(Taj)[0], beg(loc(Taj))[1], end(loc(Taj))[1]
loc(Taj)[2]	loc(Taj)[1], beg(loc(Taj))[2], end(loc(Taj))[2]

Table 4.2: Dependence in the Atlantic City example

events in the both interval 0 and interval 1. For instance, if we had left NYC during interval 0, and had not arrived during interval 1, then the probability of arriving in interval 2 is high.

If there were more intervals, then in general, arrival during an interval would depend on arrival during all previous intervals, and leaving NYC in all previous intervals. In practice, it would probably make sense to limit this somehow: for instance, whether or not we left New York 3 days ago probably doesn't bear much on our arrival time now.

That just leaves us with the fact of being at the Taj Mahal and the enabling event for the same. The enabling event depends only on the arrival event. The minute we arrive, we are at the Taj. The fact itself depends on three things: the enabling event and the clipping event in the same interval, and whether or not we were already at the Taj during the previous interval.

The discrete time net $B = (V, E)$ that results is shown in Figure 4.6. As in the definition, there is a node $v \in V$ for each $s \in \mathcal{SS}$, and a set of edges for each dependence relation in Table 4.2.

The definition of the discrete time net in (4.2) is very general. For instance, it is possible that some asets are represented less or more frequently than others. Restricting and relaxing D in various ways produce interesting classes of models. A

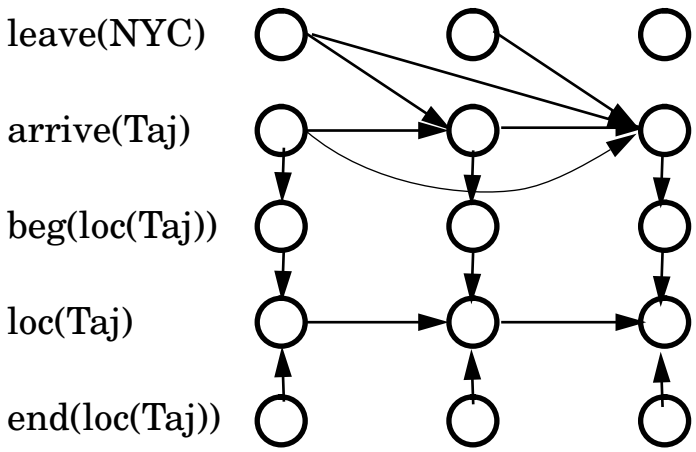


Figure 4.6: The discrete time net for the Atlantic City example.

generalization of the model may even have a different time partition for each aset. In such a model, a domain instance must take a different form. We do not explore such models here.

In a *regular discrete time net*, we do the opposite, replicating each aset for all time intervals, with the distribution functions replicating the pattern of influence, except possibly at the boundary intervals containing eti and lti. The Atlantic City example above is a regular discrete time net.

Restrictions on $\mathcal{T}$ produce regular discrete time nets with a *fixed time model* in which each time interval in $\mathcal{T}$ has a fixed constant length, and those with a *telescopic time model* in which case the interval between consecutive time points is small near the present time, but becomes gradually larger between consecutive time points in the future.

Another class of discrete time net may add restrictions in line with the ontology of facts and events developed in Chapter 2. For each fact, we add its clip and enable events, and the distribution for each fact aset must include the clip and enable events for each fact in the conditioning set. Again, the Atlantic City example had this property. Note that if in the background theory, a fact or event influences another fact or event, then the former must influence other facts and events in the latter's aset.

It is not strictly necessary to add beg(π) and end(π) for each $\varphi \in \mathcal{F}$. Even if they are present in the domain theory, it is possible to make enabling and clipping into abstract events that are not actually represented in the net. What we mean by

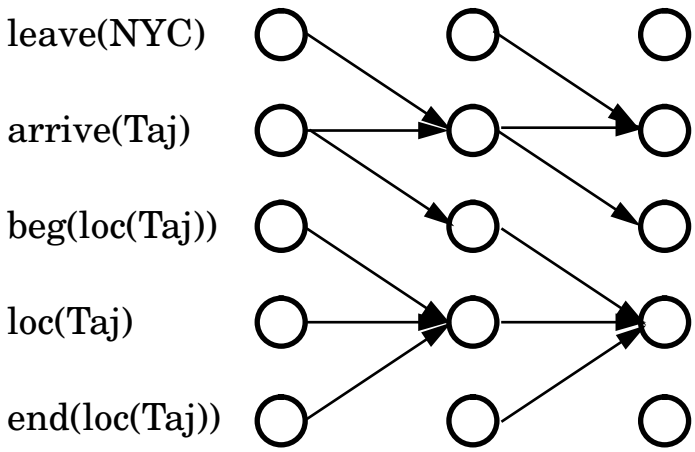


Figure 4.7: A proper Markov time net.

this is the following. Suppose that π becomes true as a result of the event α. For simplicity, assume that nothing makes π false. In the Atlantic City example, this corresponds to letting π be `loc(Taj)` and α be `arrive(Taj)`. As with the example, π depends on `beg(`π`)`, and the latter depends on α. However, it should be easy to see that in order to reason about π, we can reason about π on the basis of α, without actually thinking about `beg(`π`)` directly. In that case, π inherits directly from the parents to `beg(`π`)` and `end(`π`)`. For instance, instead of regarding X as a fact that causes Y to become true, we can consider X as evidence of Y being true.

There is an algorithm for actually performing such an optimization, through the process of *node absorption*, which is described in in Section 4.4. This can result in substantial computational savings.

We now focus on models with highly regular structure, models related to Markov processes. This type of model, the *Markov time net*, has been investigated in past work leading to this dissertation, and independently by Tatman and Shachter [Tatman and Shachter, 1990].

In a Markov time net, the conditioning set for each $\sigma \in \mathcal{SS}$ is restricted to asets either in the same or previous time interval (see Figure 4.7). When the conditioning set belongs strictly to the previous time interval, it is a *proper Markov time net*, otherwise it is a *quasi-Markov time net*. Strictly speaking, a proper Markov time net is a *proper order 1 Markov time net*. In the Markov time net pictured above, we have converted the Atlantic City into a proper Markov time net by letting influences span only across time intervals, not within.

More formally, a discrete time net is an order 1 Markov time net iff

$$\forall \iota \in \mathcal{I}, v \in V_\iota, Pa(v) = \emptyset \vee Pa(v) \subseteq V_{Pred(\iota)}.$$

A discrete time net is is quasi-Markov iff

$$\forall \iota \in \mathcal{I}, v \in V_\iota, Pa(v) \subseteq V_{Pred(\iota)} \cup V_\iota - v \wedge Ch(n) \subseteq V_{Succ(\iota)} \cup V_\iota - v.$$

Let us now formally relate the Markov time net to results in stochastic process theory. Suppose that the instantaneous state of the world can be completely specified in terms of a vector of values assigned to a finite set of boolean variables $\mathcal{P} = \{\pi_1, \pi_2, \ldots, \pi_n\}$, and suppose further that the environment can be accurately modeled as a *Markov process* in which time is discrete and the state space Ω corresponds to all possible valuations of the variables in $\mathcal{P}$. Given such a model including a transition matrix defined on Ω, we can generate a time net that enables us to compute the probability of any proposition in $\mathcal{P}$ being true during any time interval `t` given evidence concerning the values of variables in $\mathcal{P}$at various times, and do so in accord with with transition probabilities specified in the Markov model. Nunez [Nunez, 1989] has proved equivalence in expressive power between temporal Bayes nets and Markov chains. She provides a procedure which, given a Markov time net, constructs a corresponding Markov model that encodes the same information.

The reason for using the time net model rather than an equivalent Markov model is twofold. First, the time net is designed to simplify computing the answers to questions that one generally needs for applications in planning and decision support: namely, answers to questions of the form, "What is the probability of π at `t` given everything else we know about the situation?" These same answers can be computed using the Markov model, but the process is somewhat less direct.

Second, in general, the time net representation is considerably more compact than the Markov chain representation. Markov models require transition matrices with size $O(2^{2n})$, $n = |\mathcal{P}|$ (Note however that in practice, this space requirement is reduced with sparse matrix methods). The models that we conjecture are equivalent to Markov models specify how each proposition at a time interval relates to each proposition in the previous time interval. This information is then replicated as many times as we have time points that we are interested in reasoning about. If m is the maximum number of incoming arcs for any node in the time net, then the size of the resulting time net, including all of the associated probabilities necessary to complete the model, is $O(2^m kn)$, where k is the number of time intervals and $n = |\mathcal{P}|$.

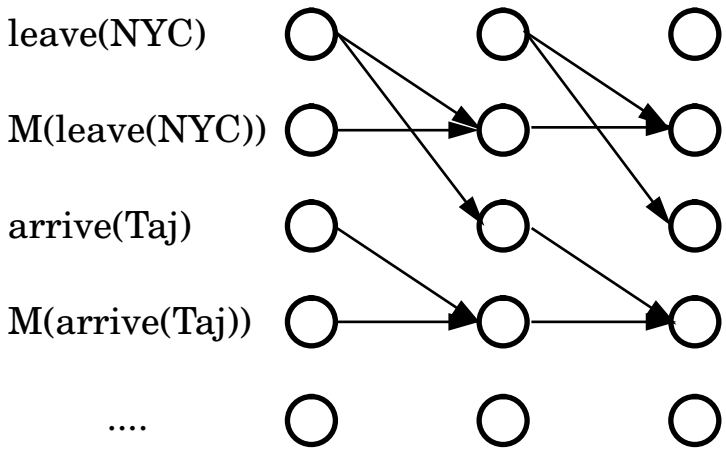


Figure 4.8: A discrete time net with memory.

There are various issues with Markov models that have been explored before. Some are modeling difficulties that come from the restriction that influences cannot span more than n intervals. For instance, at some place, it needs to be ensured that each event token ϵ sums up to at most 1 in probability over the time line. In a 1-Markov model, this is not possible, without some form of "memory". Suppose that ϵ occurred at time interval n. Then the probability that it occurs for all other time intervals should be 0. This is easy to ensure for interval $n + 1$. But after that, for all we know, ϵ has not occurred yet, so we must add a fact `epsilon-has-occurred`. `epsilon-has-occurred` has probability 1 at the interval or in the next interval after ϵ happens, and always thereafter. And `epsilon-has-occurred` makes ϵ false thereafter. This isn't surprising in some sense ("probability that X occurs given it hasn't occurred yet").

Let us now consider a type of Markov time net that adds "memory". In such a Markov time net, for each event ε, we add its *memory* `M(`ε`)`, a fact that indicates whether or not the event has already occurred. That fact must be included in the conditioning set of the aset for the event. Such a time net is depicted in Figure 4.8. In either case, as before, an event cannot be caused except in the next instant in a simple Markov (order 1) model, without defining more token types such as the fact "X became true 1 interval ago". In general, it is not impossible to have effects span more than one state transition in a Markov model, but it comes at considerable expense in the model size, by adding more fact types, *i.e.*, in terms of the Markov chain equivalent, by adding elements to the state vector.

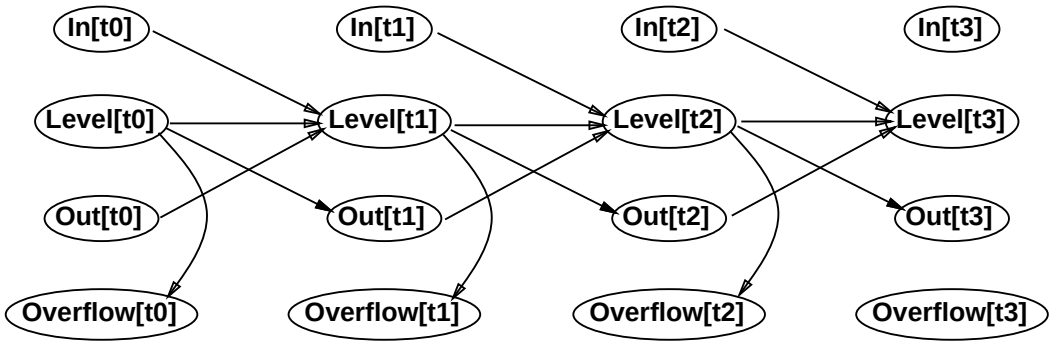


Figure 4.9: Reasoning about water tank level.

Finally, it is also possible to augment a discrete time net with nodes not directly created from domain instance tokens. Such nodes are typically added to enforce a constraint not directly expressible in the standard discrete time net framework. These include mutual exclusion conditions, and Markov n order constraints.

More useful additions include nodes that indicate the duration of facts. However, such nodes are expensive to add, clutter up the network topology, and their probability functions can be very cumbersome to specify. In domains where such information is important, it would probably be better to consider the continuous time net.

Let us now consider the extension of the discrete time net model to the case of continuous random variables. This is relatively straightforward. For a continuous random variable, the concept of the aset must be replaced with that of a continuous range, and the concept of the conditional probability table must be replaced with that of a continuous density function.

One important modeling decision with regard to continuous variables is what value to model. For a continuous variable during some finite, non-zero interval, what do we select as its value? There are a number of choices, the simplest being the value at the start, middle, and end of the interval, and the average or median value during the interval. In general, there is no right answer. It depends on the domain and on the discretization of time that has been chosen. In fact, in practice, it is likely that the discretization is chosen on the basis of the behavior of one or more of the continuous variables being modeled rather than the other way around.

As an example, we can model the water level of some tank over successive intervals of time. The level at each interval in time depends on the level in the preceding interval, and the amount of water that was flowing in and out of the tank in the preceding interval. If the water level is above a certain level, then it overflows (Figure 4.9).

One way to model the situation is by letting the water level be a random variable. Let it be a normally distributed random variable with mean μ_L and variance σ_L. Let the input also be normally distributed, with mean μ_I and variance σ_I. We assume that output is constant at c (unless there is no water). Overflow is a binary proposition that is true if water level is above a threshold θ. These dependencies are shown as edges in Figure 4.9.

The value of each random variable given its parents is given not by a conditional probabilities, but by equations:

$$\begin{aligned} \texttt{Level[t+1]} &= \texttt{Level[t]} + \texttt{In[t]} - \texttt{Out[t]} \\ \texttt{Out[t+1]} &= I(\texttt{Level[t]} > c)c + I(\texttt{Level[t]} \leq c)\texttt{Level[t]} \\ \texttt{Overflow[t]} &= I(\texttt{Level[t]} > \theta) \end{aligned}$$

Given the water level at time 0, we can predict the future levels using the time net. Essentially, this corresponds to a multivariate normal model. This is not necessary. For instance, we may have had a uniform distribution for the input variable. In general, it would be necessary to use stochastic simulation (Section 4.4.3) for inference with general continuous random variables.

4.3.2 The Continuous Time Net

A continuous time net is a Bayesian network that uses continuous valued random variables to model the time of occurrence of events of interest in the world. The continuous time net models the probability of facts and events and a host of other useful random variables based on functions of facts and events. The continuous time net is based on the *network of dates* proposed by Berzuini [Berzuini, to appear] (and apparently by Spiegelhalter, as cited in same). Berzuini's networks model only what we have called events in our terminology. We have extended this model to include facts as well as arbitrary functions of event dates and fact durations.

The continuous time net is of a different flavor and character compared to the discrete time net. Each random variable has as its state space the time line. There is a probability density over the time line itself, rather than distributions over different alternates at different intervals of time. In a continuous time net, a fact is represented by the time it begins to be true, and the time it ends being true; in other words, the times of its enabling and clipping events. In general, a continuous time net is more expressive and more compact compared with a discrete time net. There is no

Markov type restriction, and conditional influences are generally represented with simple parametric functions, rather than exponential sized tables. Not only that, such functions are more amenable to learning through experience. For this reason, the second generation probabilistic temporal database developed in Chapter 5 uses continuous time nets.

For the formal development of the continuous time net, we begin again by defining a problem instance. Let a *domain instance*, or *domain*, D be the tuple $D = \{\mathcal{F}, \mathcal{E}, \mathcal{S}, \mathcal{P}\}$. Each element has the same meaning as before. $\mathcal{F}$, $\mathcal{E}$, $\mathcal{S}$, and $\mathcal{P}$ are, respectively, the set of fact tokens, the set of event tokens, the asets, and the probability functions.

Let a *restricted domain* be a domain where the cardinality of each aset $\sigma \in \mathcal{S}$ is at most 2. The only alternates allowed for each fact and event are whether or not the fact holds, and whether or not the event occurs. Assuming that a fact and its negation do not both belong to $\mathcal{F}$, and that an event and its negation do not both belong to $\mathcal{E}$, then the cardinality of $\mathcal{S}$ is the sum of the cardinality of $\mathcal{F}$ and $\mathcal{E}$ together. For the moment, we will focus on restricted domains.

For each event $\epsilon \in \mathcal{E}$, let its *date* δ_ϵ be the unique time $-\infty \leq \delta \leq +\infty$ when ϵ takes place. By $\delta_\epsilon = +\infty$, we mean that the event ϵ never takes place. In the following, an event ϵ is often used as shorthand notation for its date δ_ϵ. For instance, in talking about the probability that ϵ occurs between time t_0 and t_1, we commonly express this as $P(t_0 \leq \epsilon \leq t_1)$, rather than $P(t_0 \leq \delta_\epsilon \leq t_1)$.

For each fact $\varphi \in \mathcal{F}$, let its *range* ρ_φ be the maximal interval $(\texttt{beg}_\varphi, \texttt{end}_\varphi)$, $-\infty < \texttt{beg}_\varphi < \texttt{end}_\varphi \leq +\infty$ over which φ holds true. Assume for the moment that each φ is *liquid*, *i.e.*, there is no break in the interval over which it is true. Thus,

$$\forall u, v \; \texttt{beg}_\varphi \leq u < v \leq \texttt{end}_\varphi \; \texttt{holds}(u, v, \varphi).$$

As with events and their dates, we commonly use a fact symbol to denote its range.

Following Chapter 2, let the *enabling event* of φ be the event of φ becoming true, and the *clipping event* of φ be the event of φ becoming false. We write down these events as $\texttt{beg}(\varphi)$ and $\texttt{end}(\varphi)$. Then the date of $\texttt{beg}(\varphi)$ is $\texttt{beg}_\varphi$, and the date of $\texttt{end}(\varphi)$ is $\texttt{end}_\varphi$.

A *complete* domain D is a domain such that for each $\varphi \in \mathcal{F}$, the enabling and clipping events $\texttt{beg}(\varphi)$ and $\texttt{end}(\varphi)$ belong to $\mathcal{E}$. We assume that all domains with regard to continuous time nets are complete. The *domain dates*, or simply *dates*, $\mathcal{D}$

of D is the set of dates for all $\epsilon \in \mathcal{E}$ for D. The *domain ranges*, or simply *ranges*, $\mathcal{R}$ or D is the set of ranges for all $\varphi \in \mathcal{F}$ for D.

The probability functions $\mathcal{P}$ of D contain a *probability density function* for each $\delta \in \mathcal{D}$ and for each $\rho \in \mathcal{R}$. Each probability density function $\pi_\delta \in \mathcal{P}$ is a function mapping from a subset of $\mathcal{D}$ and the time line to the positive reals. Let $\Gamma_\delta = \{\gamma_0, \ldots, \gamma_m\}$ be the *conditioning events* of δ, $\Gamma_\delta \subseteq \mathcal{P} - \delta$. Each π_δ is a function with the following intended semantics:

$$\pi_\delta(\gamma_0 = \xi_m, \ldots, \gamma_m = \xi_m, t) = f_\delta(t|\gamma_0 = \xi_0 \wedge \ldots \wedge \gamma_m = \xi_m)$$

or the conditional density of δ given the conditioning events. As usual, the *cumulative density function* F is defined as

$$F_\delta(t) = \int_{-\infty}^{t} f_\delta(u)du.$$

Each π_δ must be a proper density function such that

$$F_\delta(\infty) = \lim_{t \to +\infty} F_\delta(t) = 1.0.$$

Simple examples of density functions include the normal density.

$$f(t) = \mathcal{N}(\mu, \sigma, t)$$

for mean μ and variance σ. A density function may include components that act essentially like conditionals. For instance, the time somebody arrives in Grand Central Station starting from their house may depend on their mode of transport. Let time of arrival be α, the mode of transport be τ, with values *walk* and *bike*. The probability density function might be

$$\begin{aligned} \pi_\alpha(t, \tau) \quad = \quad & I(\tau = walk)\mathcal{N}(\mu_w, \sigma_w, t) \\ & I(\tau = bike)\mathcal{N}(\mu_b, \sigma_b, t) \end{aligned}$$

where $I(\cdot)$ is the *indicator function* which has value 1 if its argument expression is true, and 0 otherwise. In the above example, the probability density function will be a normal density with different mean and variable depending on the mode of transportation.

For another example, suppose that our friend Karla needs to reach London. He can either rendezvous with his friend George and hitch a flight, or he must take the train. Although Karla and George have set up a rendezvous point, the time at which

they will arrive at the meeting location is uncertain, and neither will wait for the other because it is too dangerous. Let Karla's time of arrival in London be λ, the time of his arriving at the meeting point α, and the time of George arriving at the meeting point β. Then we might represent his time of arrival in London as follows:

$$\begin{aligned} \pi_\lambda(t,\alpha,\beta) &= I(\alpha=\beta)\mathcal{N}(\mu_f,\sigma_f,t) \\ & \quad\; I(\alpha\neq\beta)\mathcal{N}(\mu_t,\sigma_t,t) \end{aligned}$$

If they manage to meet, the time of arrival is given by a normal density with mean and variance of flight time f, and mean and variance of train travel time t otherwise.

There is a special density that is often convenient for representing point events. This is the *Dirac pulse (delta) function* $\delta_a(t)$ defined as

$$\begin{aligned} \delta_a(t) &= 0 \;\forall t \neq a \\ \int_{-\infty}^{+\infty} \delta_a(t)dt &= 1 \end{aligned}$$

A Dirac pulse function can be used to represent an event that occurs at time a. As a special case, an event that never occurs can be represented as the Dirac function $\delta_\infty(t)$. For this function,

$$F(t) = \begin{cases} 1 & \text{for } t=\infty \\ 0 & \text{otherwise} \end{cases}$$

A special constraint holds with regard to the enabling and clipping events for a fact φ. The constraint is that if the clipping event for fact occurs, it must take place after the enabling event. In addition, if the clipping event occurs, then the enabling event must have taken place. The easiest way to implement this constraint is to put one in the conditioning set of the other, and encode it within the density function.

For instance, for a fact φ, let the probability function for the date of the clipping event $\texttt{end}_\varphi$ be dependent on the date of the enabling event $\texttt{beg}_\varphi$. Then $\pi_{\texttt{end}_\varphi}$ must be a function such that

$$\pi_{\texttt{end}_\varphi}(\texttt{beg}_\varphi,\ldots,t) = \begin{cases} 0 & \text{if } t < \texttt{beg}_\varphi \\ \ldots & \text{otherwise} \end{cases}$$

Suppose that a fact φ became true at time u, and we wish to know the probability that it will hold true through at least until v. For instance, given that a bee was born on Memorial Day, what is the likelihood that it will live at least until Labor Day? For φ, the probability is given by the function:

$$\sigma_\varphi(u,v) = P(\texttt{end}_\varphi > v | \texttt{beg}_\varphi = u)$$

σ, which could also be written as

$$\sigma_{\varphi}(u,v) = P(\texttt{holds}(u,v,\varphi)|\texttt{occ}(u,\texttt{beg}(()\varphi))).$$

This is the *survivor function*, as we saw in Chapter 2.

Each density function $\pi_{\varphi} \in \mathcal{P}$ is a bivariate density mapping from a time interval to the real line. The intended semantics of π_{φ} is

$$\pi_{\varphi}(u,v) = f(\texttt{beg}_{\varphi} = u \wedge \texttt{end}_{\varphi} = v).$$

There are instances where instead of the density function, its related form, *hazard function* is more appropriate. The hazard function comes from survival analysis. In survival analysis, the chief concern is reasoning about lifetime of facts, much as we are concerned with. The hazard function of a *lifetime* T is defined as:

$$h_T(t) = \lim_{\Delta \to 0} \frac{P(t \leq T < t + \Delta t | t \leq T)}{\Delta t}$$

and equivalently as

$$h(t) = \frac{f(t)}{1 - F(t)}$$

The hazard function gives the probability that a fact fails soon after time t, given that it has lasted at least t. The hazard and the density are also related in their cumulative form. Let the *cumulative hazard* be

$$H(t) = \int_0^t h(u)du$$

Then

$$\begin{aligned} F(t) &= 1 - e^{-H(t)} \\ & and \\ F^{-}(u) &= H^{-}(-\ln(1-u)) \end{aligned}$$

Furthermore,

$$f(t) = h(t)e^{-H(t)}$$

The hazard representation is often natural, particularly for specifying the survivor function. If the hazard rate is constant, then the survivor function has exponential decay.

$$\begin{aligned} h(t) &= k \\ H(t) &= kt \\ f(t) &= ke^{-kt} \\ 1 - F(t) &= e^{-kt} \end{aligned}$$

More interesting cases involve, for instance, increasing hazard. The hazard for the lifetime of a light bulb may be an increasing function of time. Roughly speaking, in this case, an increasing hazard would mean that over time, the number of light bulbs that fail at any given time increases the longer the light bulbs have been burning.

$$h(t) = kt$$

As with the density, the hazard may include conditional components depending on conditioning events. Such a model corresponds to the proportional hazards model of [Cox, 1972; Aalen, 1989].

Given the above, we are now ready to define the continuous time net.

DEFINITION 4.3
A continuous time net for D is a Bayesian belief network $B = (V, E)$. V consists of one node for each fact in $\mathcal{F}$ in D and each event in $\mathcal{E}$ D. E consists of a set of edges for each $\pi \in \mathcal{P}$ in D as follows. For each $\pi_v \in \mathcal{P}$, add the edge (γ, v) for each $\gamma \in \Gamma_\sigma$. E is the union of all such edges.

Again, the above definition gives a very general specification for a continuous time net, and that in a relatively restricted form. B must be acyclic, again, in order that it be a proper Bayesian network. There are a few things to note about the continuous time net as defined in this fashion.

First of all, with regard to the probability of facts, note that the quantity that we often wish to compute is not the density of its range, but the *probability mass function* giving the probability

$$P(\mathtt{beg}_\varphi \leq u \wedge v \leq \mathtt{end}_\varphi) = \int_{-\infty}^{u} f_{\mathtt{beg}_\varphi}(x) \int_{v}^{+\infty} f_{\mathtt{end}_\varphi}(y|\mathtt{beg}_\varphi = x)dydx$$

In the actual implementation of time nets, such probability is directly recovered at each node representing the range of a fact.

A related thing to note is that in the definition of probability functions $\mathcal{P}$, only dates were allowed in the conditioning set. Thus, range nodes are not used for anything other than for range estimates themselves. As we see when discussing computation, this means that if we know beforehand that we will not query the probability of some fact, then the range node for the fact can be removed safely without affecting the topology. There is no problem however, for implementing probability functions that encode knowledge of the form "if event ϵ occurs while φ holds, then the consequent event β occurs". This is simply encoded as a function that takes the enabling

and clipping events as part of the conditioning set. The condition " while φ holds" is replaced by "$\texttt{beg}_\varphi$ has occurred and $\texttt{end}_\varphi$ has not yet occurred".

Let us now show an example of a domain instance and its continuous time net. We will consider how the Atlantic City example of Section 4.3.1 is encoded in a continuous time net. As previously, there are two events `leave(NYC)` and `arrive(Taj)`. There is also the fact `loc(Taj)` and the enabling and clipping events `beg(loc(Taj))` and `end(loc(Taj))`. `arrive(Taj)` follows `leave(NYC)` by a certain amount of time, whereas `beg(loc(Taj))` is caused immediately by the arrival event.

The domain instance D consists of the following.

$$\begin{aligned}\mathcal{F} &= \{\texttt{loc(Taj)}\}\\ \mathcal{E} &= \{\texttt{leave(NYC)}, \texttt{arrive(Taj}, \texttt{beg(loc(Taj))}, \texttt{end(loc(Taj))}\}\end{aligned}$$

Again, the asets $\mathcal{S}$ in this case contains binary asets for each fact and event above, with the same name. The dates $\mathcal{D}$ consist of one date for each event, and the ranges $\mathcal{R}$ consists of one range for each fact, as before. Let the probability functions be given as follows.

$$\begin{aligned}\pi_{\texttt{leave(NYC)}}(t) &= \mathcal{N}(\text{noon}, 0{:}15, t)\\ \pi_{\texttt{arrive(Taj)}}(t) &= \mathcal{N}(\Delta(\texttt{leave(NYC)}) + 2{:}00, 0{:}45)\\ \pi_{\texttt{beg(loc(Taj))}}(t) &= \delta_{\Delta(\texttt{arrive(Taj)})}(t)\\ \pi_{\texttt{end(loc(Taj))}}(t) &= 0\end{aligned}$$

$\mathcal{N}(\mu, \sigma, t)$ is the normal density, with mean μ and deviation σ, for point t. For brevity, we let 0 be a function that is 0 everywhere above.

The continuous time net for D is shown in Figure 4.10. The time net contains a node for each event, and one node for the single fact `loc(Taj)`.

An important question arises when considering a domain instance and a continuous time net. Given an arbitrary domain instance that would not result in the creation of a cyclical network, is the result always a proper time net? Berzuini has shown results pertaining to the types of dependencies that can be represented in a network composed solely of events [Berzuini, to appear]. The short answer is that we cannot build time nets easily for arbitrary domain theories.

To build a continuous time net for domains that are not restricted domains, care must be taken with the probability functions. In general, there are no simple ways

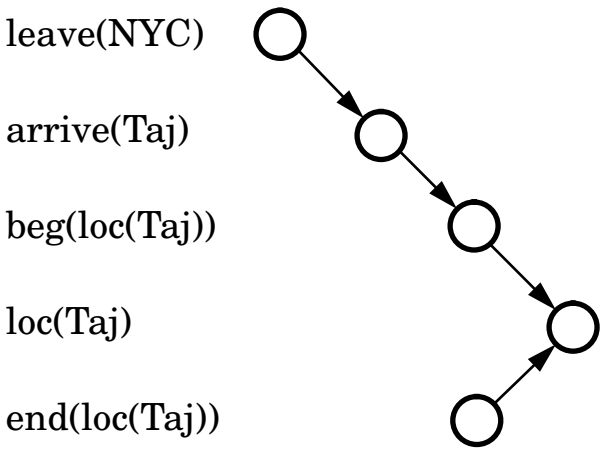


Figure 4.10: The continuous time net for D.

for ensuring that outcomes remain exclusive, beyond specifying consistent densities. For example, suppose that two events α and β are mutually exclusive and exhaustive. Given the fact γ, α has a high probability of occurring between within a half hour of γ becoming true. Thereafter, β has a high probability of occurring. For simplicity, assume that γ stays true forever once it become true. Given that α and β are mutually exclusive and exhaustive, the sum of the probability that they occur is 1.0. In general, it is difficult to enforce this condition.

There are two solutions. One is to have *mutual exclusion nodes*, and the other is to have *oracle nodes*. A mutual exclusion node has as its parents a set of mutually exclusive and exhaustive events. The probability of the node is 1 iff one of the events has a date less than infinity. Otherwise it is 0. The node is predefined to have probability 1. The other solution is to have an oracle node, that determines which event occurs first. Neither is completely satisfactory, but because of the Berzuini's result, some accommodation is necessary to represent many types of dependencies.

A continuous time net can be augmented with nodes representing random variables that are the result of functions and relations of dates and ranges. Such random variables are generally atemporal. We have seen that the density function for a date can include temporal relations. For example, the density of an event γ may depend on whether or not the event α precedes another event β. It is also possible to represent such relations directly as random variables. For instance, we may choose to represent

$$P(\alpha < \beta | \theta)$$

in terms of a node ζ, where θ represents "all known evidence". Associating such a

fact with a point or interval of time makes little sense. Either α precedes β or not, and it does depend on any interval of time.[2] The probability function for such a node would be given by

$$\pi(\zeta|\alpha,\beta) = I(\alpha < \beta)$$

where I is the *indicator* function

$$I(\varphi) = \begin{cases} 1 & \text{if } \varphi \\ 0 & \text{otherwise} \end{cases}$$

Another example of a common temporal relation might be probability that a fact φ holds until at least some deadline τ.

$$P(\mathtt{end}_\varphi > \tau|\mathtt{beg}_\varphi = \beta, \theta)$$

Such a probability is useful during planning.

Abstractly, we could define a node for each such temporal relation used in a density function. In practice, this need not be done, however. In general, adding a node for such temporal relations only adds overhead, unless we know beforehand that the information is being sought.

Temporal relations are not the only type of extensions that are possible in a continuous time net. In general, we may allow, recursively, arbitrary functions of dates and ranges, and functions of those functions, etc.. For instance, it is possible to represent the lifetime Λ of a fact as a random variable. As before, the lifetime of a fact is naturally computed easily as the difference of the dates of its beginning and end. For instance, for a fact token representing the occurrence of a traffic jam τ, we would have the nodes $\mathtt{beg}_\tau$ and $\mathtt{end}_\tau$. The lifetime Λ has the conditional density

$$\Lambda_\tau(t|\mathtt{beg}_\tau = \beta, \mathtt{end}_\tau = \epsilon) = \delta_{\epsilon-\beta}(t)$$

This is true in general for all fact tokens τ. Nothing prevents the creation of nodes that represent the difference of arbitrary event dates. For instance, instead of the overall probability that event α precedes event β, our interest may lie in by the density function of how much α precedes β

$$f(\beta - \alpha).$$

Again, such a density would be easy to specify given the event dates α and β.

[2]Unless there is quantification involved. Even then, it only makes sense to consider something like "the probability that some event of type x precedes some event of type y during interval I".

Another example might involve the sum of durations. Resource utilization and allocation is often an important measure in economic decision making. One example involves the scheduling of commercial aircraft. An airline wishes to neither underutilize aircraft, since it would waste money buying aircraft it hardly ever uses, nor to overutilize aircraft, for safety considerations. Thus it may pay to sum the likely durations for which it will be used over some period of time. If we plan some $\{\tau_0, \ldots, \tau_k\}$ trips for the aircraft, then the total time the aircraft is used is given by

$$f(\Lambda(\tau_0) + \ldots + \Lambda(\tau_k)).$$

The node for this sum would have the density of the sum time that the aircraft is used. Such a function is a *deterministic* function of its arguments, because it is completely determined by the value of the arguments.

Having added nodes representing simple functions of dates and ranges, we may recursively add nodes representing functions of those. An example is the cost associated with the duration of some fact or the probability of some temporal relation. For instance, suppose that there is a task τ with a deadline δ. Assume that τ is a task represented as a point event (or alternatively, assume that τ is the end point of a task represented as a fact). The cost might be given by a function of the binary relation $\tau < \delta$

$$cost(\tau, \delta) = \begin{cases} 0 & \text{if } \tau < \delta \\ +1000000 & \text{otherwise.} \end{cases}$$

or by the difference $\delta - \tau$, how much before or after the deadline τ is expected to take place:

$$cost(\delta - \tau) = \begin{cases} 0 & \text{if } \delta - \tau > 0 \\ e^{-k(\delta - \tau)} & \text{otherwise} \end{cases}$$

A node representing the cost has associated with it the cost density. The total expected cost is the integral over all costs of the product of the cost and its density:

$$E(cost(\delta - \tau)) = \int_{-\infty}^{+\infty} cost(\delta - \tau) f(\delta - \tau) d\tau$$

As we shall see in later chapters, such expected cost is used in decision theory for choosing actions.

4.3.3 Discussion

The relative merits of continuous time net versus discrete time net are as follows. Generally speaking, both have advantages, with the continuous time net appearing

to have fewer limitations with one notable exception.

The discrete time net offers a conceptually simple model when we can reason about a domain easily as a sequence of snapshots of system state. In a discrete time net, we basically model a domain as a set of system variables at a point in time, replicate that set of variables across a range of points in time, and use edges spanning time points to represent influences. The evolution of the system over time is thought of in terms of how one snapshot leads to the next, and successively to points further on. It is especially easy to reason about the evolution of continuous variables over time with this method.

The continuous time net also offers a conceptually simple model, in this case when we can reason about a domain easily in terms of a set of distinct event occurrences and how they change system state. Each system state attribute and event is modeled as a single vertex, and edges model the relationship between them. It is especially easy to reason about exactly when events take place, and how long facts are true for with this method. Furthermore, there is a simple mapping from logic of lifetimes concepts to continuous time net concepts. It is also possible to use a wide class of parametric distributions to model event and fact densities.

The flip-side of the discrete time net approach is that we are *committed* to representing all time points that are necessary to model a problem, and this potentially involves a large total number of variables. If the interval between snapshots is fixed, then a large number of snapshots may be needed to reason about longer durations. If the interval is not fixed, then there is a necessity to gather a large body of statistics. If the model sticks to a strict Markov chain, where edges can only bridge successive time points, then it is difficult to model duration of facts, event occurrences, and general parametric distributions. Although it is possible to amend this by either allowing general n-Markov influences or by adding "memory", this results in considerable model complexity, both in terms of overhead of representing the model, and also in visualizing and manipulating the model. This is often a considerable burden on model builders and inference algorithms.

The flip-side of the continuous time net approach is that although it avoids many of the discrete time net's disadvantages, it generally commits to reasoning in terms of distinct events. If, for example, somebody may enter or leave a room multiple times, then generally, with the continuous time net approach, it is necessary to model all distinct instances of entering and leaving the room. By contrast, a discrete time net

can abstract away the instances and model an event type (and thereby lose information about instances such as duration). Continuous variables are often considerably simpler to model with the discrete method.

Continuous time net models are considerably more flexible for dynamic domains. Whenever a new event is added, only one vertex needs to be added, and whenever a new fact is added, only three vertices need to be added. By contrast, with the discrete time net, each such vertex must be replicated over all time points. Although some of the memory requirements can be similar for both approaches because continuous time net vertices must discretize a density into a vector of time points, there is still considerable overhead for each vertex with the discrete time net. Furthermore, with the continuous time net, the discretization can be easily changed for different situations and different variables, even at run-time. This is because the sampling functions don't change on the basis of the discretization. By contrast, this is difficult for discrete time nets, requiring the maintenance of many different probabilities for different discretizations.

It is possible to circumvent almost all disadvantages of the discrete time net by allowing influences to span multiple time points, by adding special types of vertices (for instance, to model temporal relations), and by relying on intelligent model construction. However, this makes the models far more complex, and merely shifts the burden to the model builder, both human and machine. It remains to be seen how an intelligent model builder can choose the most appropriate models on the basis of a general purpose domain knowledge.

4.4 Computation

For the rest of this chapter, we explore computation in time nets. A time net represents knowledge about time, probability, and dependence as knowledge about a set of random variables. The main use of the time net is for answering queries about probabilities of facts and events over time. When a time net is created, only the marginal distribution of random variables associated with root nodes are already known, perhaps along with some evidence about the value of some other random variables. Marginal distributions for all other nodes must be computed on the basis of conditional distributions and known marginal distributions. As time nets are simply Bayesian networks with a special interpretation of its random variables, the

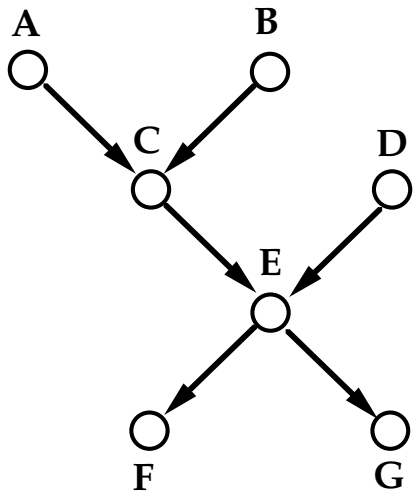


Figure 4.11: A polytree

algorithms employed for such computation are just Bayesian network algorithms.

We define the problem of *probabilistic inference in a Bayesian network* [Cooper, 1990] as the computation of a consistent distribution for each node $v \in V$ of a Bayesian network B, possibly given evidence on a subset of the nodes $K \subseteq V$. Evidence corresponds to the value of a random variable being known unconditionally, *i.e.*, the probability of one of its alternates is 1. We also defined this problem as *evaluating a Bayesian network* previously.

A standard method suggested by probability theory for the probabilistic inference problem would be to store or compute a joint distribution that ranges over V. The distribution for any random variable $v \in V$ is computed by marginalizing (summing) over all the other variables $V - \{v\}$. Needless to say, this method is impractical for all but the smallest models. A key idea behind the use of Bayesian networks for probabilistic inference is to exploit the network topology in a combination of distributed local computation and global constraint propagation to avoid manipulating a huge joint distribution.

A number of such algorithms have been developed; it turns out that the class of algorithms that can be used, and indeed the computational complexity of the probabilistic inference task, in a Bayesian network critically depends on the topology of the graph. The major topologies of interest are *polytrees* and dags with (undirected) cycles.

For polytrees, Kim and Pearl [1983] have devised a simple constraint propagation

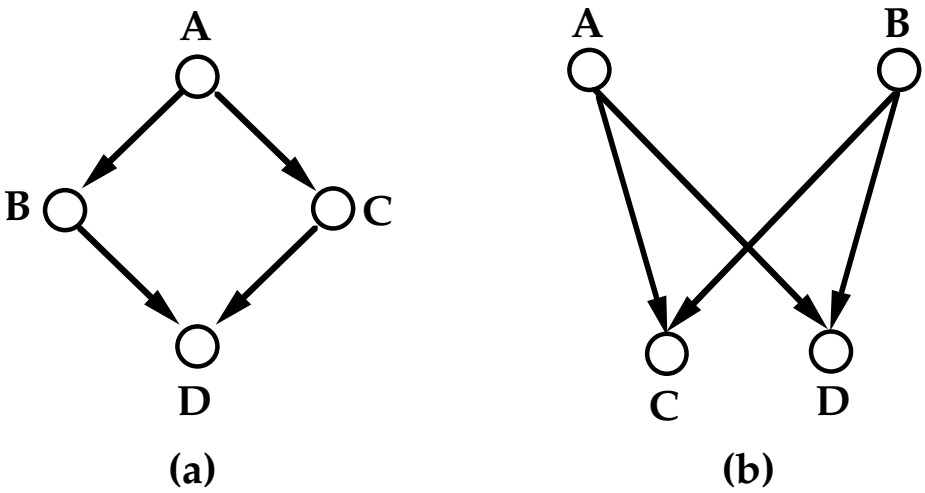


Figure 4.12: Multiply-connected networks

scheme that runs in time polynomial in the diameter of the tree. In this algorithm, each node sends messages comprising its probability distribution to its neighbors. The tree-like topology ensures that the probabilities propagated are consistent across the network. Peot and Shachter [1991] have recently improved the algorithm so that it takes only linear time, of order $O(2n)$, where n is the number of vertices $|V|$.

By contrast, Cooper [1990] has shown that the problem of probabilistic inference in multiply-connected networks is NP-Hard. Unfortunately, many dependency models are multiply-connected. In particular, this is true of our discrete time nets. A model is multiply-connected whenever two nodes share at least two ancestors or descendants combined. Examples include "diamond" shaped networks (see Figure 4.12(a)) as well as the case when two propositions both influence two other propositions (see Figure 4.12(b)).

Not only is probabilistic inference NP-hard in multiply-connected networks, the simple Kim-Pearl algorithm breaks down in the presence of loops. In probabilistic inference, care must be taken that evidence from any particular random variable is not counted more than once in the other random variables. This is easy in a polytree since no two nodes are connected together by more than one path. The Kim-Pearl algorithm exploits this by assuming that for any given node, information about all parts of the network beyond any one neighbor is summarized in that neighbor and only that neighbor. With the Kim-Pearl algorithm, a node does not need to know what lies beyond its neighbors; it only needs to inspect its direct neighbors to update

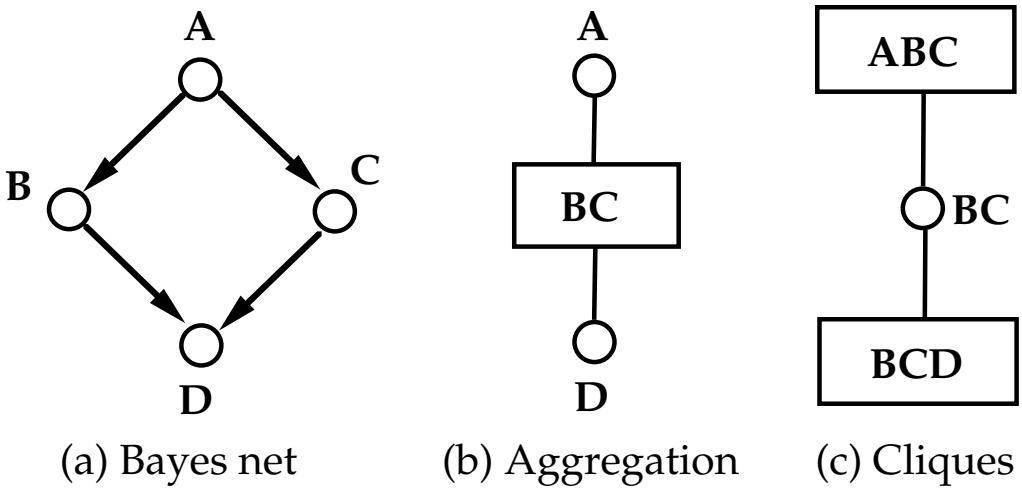


Figure 4.13: Node Clustering

its own state. Unfortunately, this assumption does not hold in multiply-connected networks. For example, in the diamond shaped example of Figure 4.12(a), both B and C might try to propagate any effects of change A to D, which means that D gets too much information.

A large number of algorithms have been developed for inference in multiply-connected networks. There are four main classes of such algorithms. The first two are exact algorithms, and the other are approximation algorithms. Due to the NP-Hardness result, all these algorithms have some facet which may undergo exponential blowup.

(1) *Conditioning*. In multiply-connected networks, some pieces of evidence may render the network effectively singly-connected. For example, in the previous diamond shaped network, if the top node is instantiated to some value, we can first propagate that effect to its children, and then pretend as if the top node didn't exist. Then the Kim-Pearl or Peot-Shachter algorithm can be applied. For any given network, a set of nodes that collectively render a multiply-connected network singly-connected in this fashion is a *cutset* of the graph. Pearl's method of cutset conditioning [Pearl, 1986] takes such a cutset of a network, instantiates the nodes in the cutset to all possible values, and computes the weighted average of the posterior beliefs derived by applying Kim-Pearl to each instantiation. Again, using the improved algorithm due to Peot and Shachter is simpler and more efficient than the original algorithm.

(2) *Clustering*. Clustering algorithms form a tree of aggregate node sets out of the

nodes of a multiply-connected network. The state space of each aggregate node set is the cross product of the nodes in the set. The distribution for each node is derived by marginalization of the/an aggregate node set that it belongs to. There are two distinct classes of clustering algorithms. Pearl's method of node aggregation [Pearl, 1988] forms a DAG out of a particular partition of nodes into clusters, and then applies the Kim-Pearl algorithm (see Figure 4.13(b)). By contrast, methods by Lauritzen and Spiegelhalter [1988] and by Jensen and others [1990] form aggregates out of the cliques of the decomposition [Lauritzen *et al.*, 1984] of the original network. The cliques are then linked either in an undirected tree or chain over which an efficient propagation scheme similar to Kim-Pearl can be applied (see Figure 4.13(c)). Whereas Pearl's node aggregation needs to propagate whole aggregate distributions to neighboring aggregates, clique based methods communicate only via the intersection of cliques.

(3) *Sampling* [Henrion, 1988a; Pearl, 1987; Chavez and Cooper, 1989; Berzuini *et al.*, 1989]. These are variants of Monte Carlo approximation. Monte Carlo schemes have nice conceptual properties, and they are the only general algorithm for continuous random variables. Therefore, for continuous time nets, these are the only algorithms considered. Such algorithms had problems with convergence in the past, but recently, these have been ameliorated [Fung and Chang, 1989; Shachter and Peot, 1989], and they have seen renewed interest [Shwe and Cooper, 1990].

(4) *Bounding*. These algorithms bound the probabilities on items of interest through heuristic search. Cooper [1984] and Henrion [1988b] have heuristic search schemes with the aim of finding the most probable conjunctive hypothesis over an entire space of possible hypotheses. Horvitz and Suermondt's *bounded cutset conditioning* [Horvitz *et al.*, 1989] is reportedly an application of heuristic search to Pearl's method of cutset conditioning.

4.4.1 The Clique Tree Algorithm

The clique (junction) tree algorithm, also known as the Bayesian belief universes algorithm, is due to Jensen, Lauritzen, Olesen, and Andersen [Jensen *et al.*, 1990; Jensen *et al.*, 1988][3]. The algorithm is closely related to another clique based algorithm, the somewhat better known Lauritzen-Spiegelhalter algorithm [Lauritzen and Spiegelhalter, 1988]. The clique tree algorithm was developed originally for a

[3]The problem of propagation of probabilities in networks with loops also arises in genetics, and it turns out that a solution similar to the clique tree algorithm had been found earlier in that context[Cannings *et al.*, 1978]. However, these results were not generally known until recently.

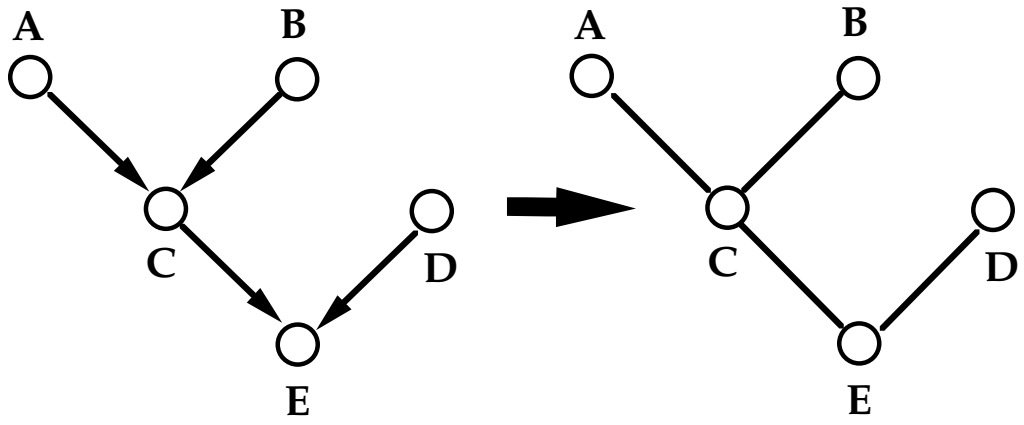


Figure 4.14: Bayesian network to Markov field

medical diagnosis application [Olesen *et al.*, 1989].

The basic idea of the clique tree algorithm is to transform a multiply-connected Bayesian network into a tree of clusters of nodes. The clusters and the tree are fashioned in such a manner that an efficient propagation scheme results in a consistent joint distribution.

The transformation consists in converting the directed Bayesian network graph into a undirected graph that is a Markov random field. This undirected graph must be a *chordal graph*, a graph that has no cycles of length 4 or more without a chord bisecting it. We define distribution functions on the cliques of the chordal graph and the cliques are joined in a tree. This is the *junction tree*. Chordality in the Markov graph is a necessary and sufficient condition for guaranteeing consistency in computations with the junction tree [Jensen, 1988]. Changes to clique distributions are propagated to neighboring cliques through the intersection of the cliques, in an operation called *calibration* or *absorption* [Jensen *et al.*, 1990]. The clique intersections themselves are called *separating node sets* or *sepsets* for short. Beliefs in individual nodes are obtained by marginalizing the clique distributions.

Let us now describe in closer detail the steps required to transform a Bayesian network into a junction tree. The Bayesian network graph is first made undirected, by changing directed arcs into simple undirected links. Then every pair of parents of every non-root node of the original directed graph are linked together ("married") in a step called *moralizing the graph* (see Figure 4.15).

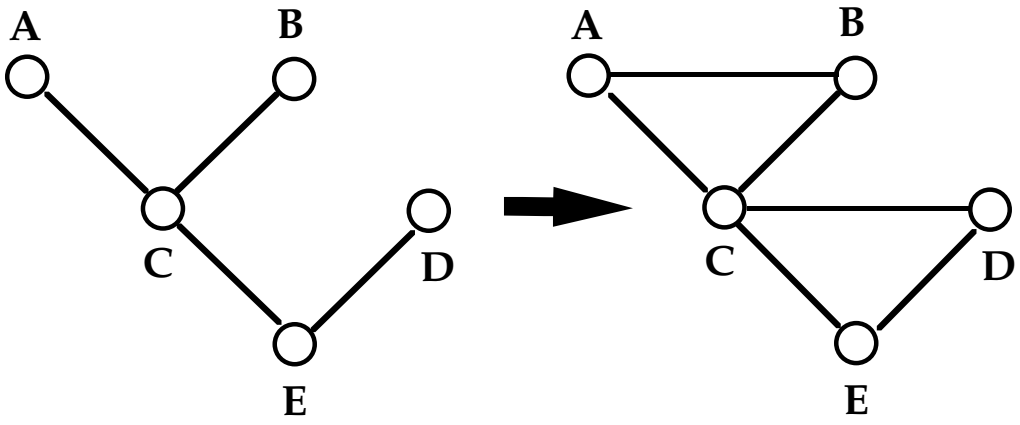


Figure 4.15: Moralizing the graph

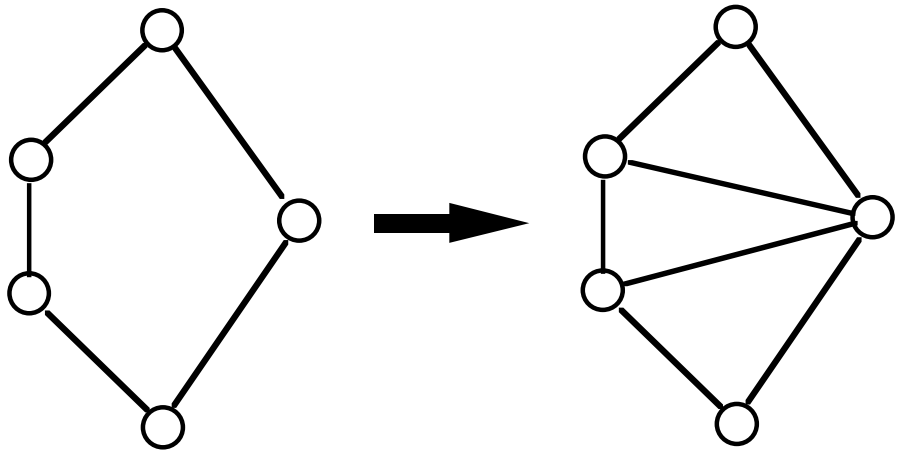


Figure 4.16: Filling-in the graph

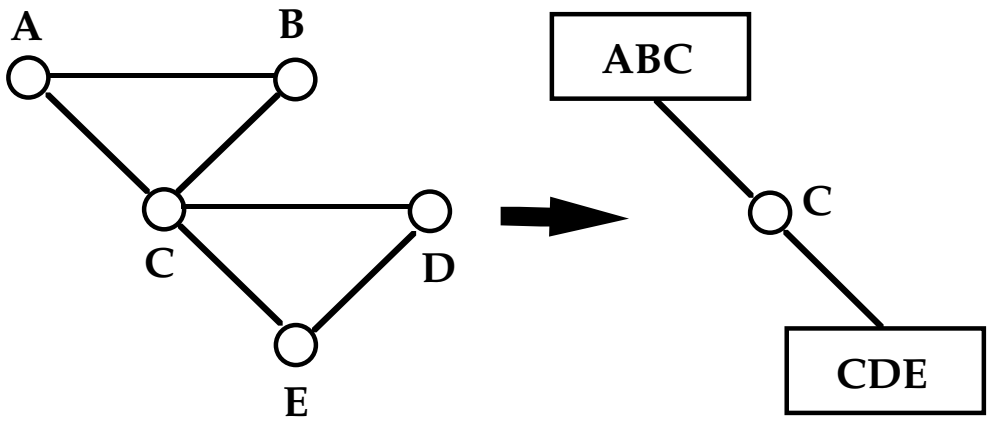


Figure 4.17: Forming the junction tree

Next, the graph is *filled-in* as necessary to make it chordal (see Figure 4.16). We will address appropriate fill-in methods shortly. The junction tree is then formed from the cliques of the tree. The cliques and the junction tree may be found by any convenient method. Jensen shows [1988] that any maximal spanning tree of the cliques of the chordal graph will be a junction tree, and suggests finding this maximal spanning tree. Our method exploits the numbering obtained by a maximum cardinality search [Tarjan and Yannakakis, 1984] that we employ in verifying the fill-in, and we form the junction tree utilizing the join tree algorithm in [Pearl, 1988] (see Figure 4.17).

In the junction tree, both cliques and sepsets have distribution functions, ϕ_C and ϕ_S. They are not probability distributions, as they are not required to be normalized (sum to 1). They are only required to have correct proportionalities. In general, this will result in substantial computational savings.

The calibration operation propagates from one clique to another via the sepset that links the two cliques. For cliques C_1 and C_2, their sepset $S_{1,2}$ contains the nodes $C_1 \cap C_2$. The *residuals* at C_1 and C_2, R_1 and R_2 will be defined to be $C_1 - S_{1,2}$ and $C_2 - S_{1,2}$, respectively. To calibrate C_2 to C_1, first of all, ϕ_{C_1} is marginalized over R_1, to create the *posterior distribution* for $S_{1,2}$, $\phi'_{S_{1,2}}$. As we compute this posterior distribution, we must save the prior distribution $\phi_{S_{1,2}}$ somewhere temporarily. Once $\phi'_{S_{1,2}}$ is computed, it is divided by $\phi_{S_{1,2}}$ element by element. This essentially corresponds to computing the posterior / prior odds. This odds distribution is then propagated to C_2, and $S_{1,2}$

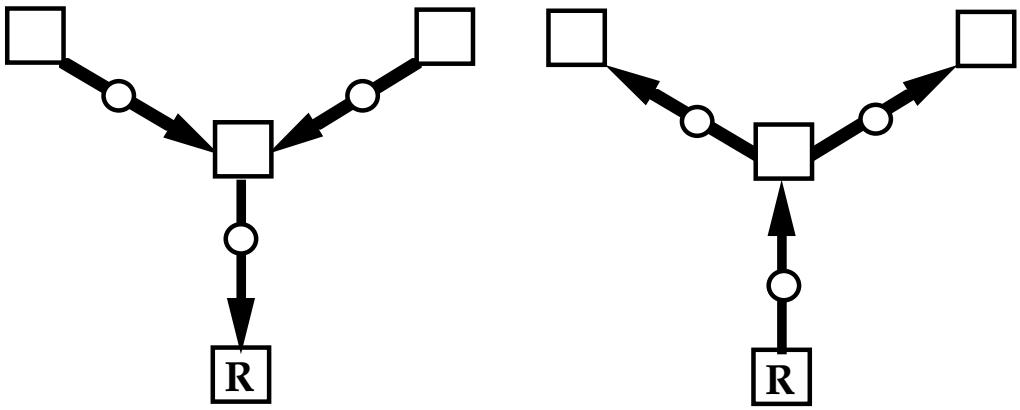


Figure 4.18: Propagation in junction tree

updates its distribution to be the posterior it received from C_1.

At C_2, the odds is multiplied into the distribution for C_2, ϕ_{C_2}. This product is performed as follows: for each point ω_{C_2} in clique C_2's state space Ω_{C_2}, we multiply the number in the distribution for C_2 by the element of the odds distribution corresponding to the element $\omega_{C_2} - \omega_{R_2}$, where $-$ is the set subtraction operation.

For tree propagation, two recursive operations, *collect* and *distribute* are defined. Both are recursive single-pass operations. We will refer to the original clique that either operation is called as the *root clique* C_R. In the collect operation, C_R calibrates to each of its neighbors, after the neighbors have collected from their other neighbors. Thus, when collect is invoked at C_R, it is recursively invoked through the whole tree until the leaves are reached. Then a calibration sequence rolls back through the tree until it reaches back to C_R (see Figure 4.18(a)).

Distribute is the inverse analogue operation. When distribute is invoked at C_R, all its neighboring cliques calibrate to it, and then distribute is invoked at those neighboring cliques in turn. It terminates when all the cliques which are leaves relative to C_R calibrate to its (single) neighbor. Thus, the calibration sequence fans out to the leaves, rather than roll back in (see Figure 4.18(b)).

To initialize a junction tree, first, the clique distributions are initialized from their constituent node distributions. Then, collect and distribute are invoked, in that order, at some random clique. This two pass operation is sufficient to put the whole network in a consistent state.

In probabilistic inference, when evidence is received, if the nodes for which there is evidence belong to a single clique, then invoking distribute at that clique suffices to update the whole network. When there is evidence in more than one clique, then a collect / distribute sequence must be performed as with tree initialization.

4.4.2 Analysis of Clique Tree Algorithm

The complexity of the clique tree algorithm is given as follows. Let C_i be clique i, and Γ_i the cost at C_i each time it is involved in a calibration operation. The adjacent cliques A_i of C_i is defined as all of C_i's neighbors. Then the complexity for each distribute and collect is

$$\sum_{C_i \in C} |A_i| \times \Gamma_i. \tag{4.1}$$

Note that we may regard each sepset as an edge of the clique tree. Thus, the sum of size of all the neighborhoods $\sum_{C_i \in C} |A_i|$ is the number of edges times two. Since the number of edges is $|C| - 1$, if we let $\Gamma_{\max} = \max_i \Gamma_i$, then the complexity is

$$O(2|C| \times \Gamma_{\max}) \tag{4.2}$$

The calibration operation takes time proportional to the state space at each clique $\Omega_{C,i}$. If $|\Omega_{N,i}|$ is the state space size of node i, then

$$\Gamma_i = O(|\Omega_{C,i}|) \tag{4.3}$$

$$= O(\Pi_{N_j \in C_i} |\Omega_{N,j}|) \tag{4.4}$$

If let $\Omega_{N,\max} = \arg\max_i |\Omega_{N,i}|$, and $S = \max_{C_i \in C} |C_i|$, then

$$\Gamma_{\max} = O(|\Omega_{N,\max}|^S) \tag{4.5}$$

Thus the total complexity can be written also as

$$O(2|C| \times |\Omega_{N,\max}|^S) \tag{4.6}$$

So the algorithm is linear in the number of cliques and exponential in the size of the largest clique [4].

Thus small clique size becomes critical in keeping the computational cost down. Note that for any given junction tree, there is a lower bound on the maximum clique

[4] In this analysis, we have ignored the calibration cost at sepsets as it is linear in the number of nodes in each sepset; it is dominated by the calibration cost at cliques, which we have seen to be exponential.

size. The initial graph moralizing step ensures that a node and all its parents in the original DAG form a clique. If a node has 100 direct causes, then given the current framework, there is no recourse but to have a clique with 101 nodes. Which is, of course, intractable not only for the clique tree algorithm, but for any current implementation of a probabilistic inference scheme.

At any rate, clique size also depends critically on the fill-in applied to make the graph chordal. Thus we would hope to apply optimizations at this phase. This is particularly attractive since for a given domain model, it is necessary to compute the fill-in only once. Thus it is a fixed amortizable cost.

There are some very efficient algorithms for computing the fill-in, such as the maximum cardinality algorithm by Tarjan and Yannakakis [1984], which runs in time $O(n+e)$, where $n = |N|$ and $e = |A|$. However, this algorithm has no optimality properties, such as having the smallest number of edges added in the fill-in. Yannakakis has proved that this latter problem is NP-complete [Yannakakis, 1981].

Our optimization problem consists in minimizing the sum in (4.1). We conjecture that this problem is also NP-complete. Because of the exponential factor, a good heuristic is to minimize the quantity $\Gamma_{\max}$.

Jensen [1990; 1990] has developed a greedy heuristic algorithm for the first problem. The fill-in is given by the following *elimination order* ([Rose *et al.*, 1976] is a good reference for elimination orders):

Step 1. Eliminate any node whose neighborhood forms a clique.

Step 2. Eliminate the node that would produce the smallest size clique.

The algorithm runs in $O((ne)^2)$. The algorithm produces good fill-ins, but as the number of vertices grows, the square factor begins to dominate. In experiments that we have run, a small network would take only seconds to triangulate, but a network of 6000 nodes would take 10 hours to triangulate.

We discovered that the first step of this algorithm takes almost all of the time, especially for larger networks, and furthermore, that removing the first step makes no difference: the algorithm produces exactly the same triangulations. When we removed the first step, the 6000 node network took only a couple of minutes to triangulate. Whereas previously, our implementation ensured that the results of triangulation were periodically saved so that the program could be interrupted, it turned out that this was unnecessary. The heuristic was fast enough that we might as well just triangulate

each time – it was faster to do that, than to save the result of the triangulation to file, or to read it back.

As it turned out, this simpler algorithm was previously discovered by Rose at least as early as 1976 [Rose *et al.*, 1976]. So although we improved Jensen's algorithm, it had been discovered long before Jensen. At any rate, the complexity of the Rose algorithm is $O(n + e')$, where e' is the number of edges, plus the fill-in.

Let us finally turn our attention to the important question, "How well does this algorithm do for time nets?". Let us note, first of all, that the clique tree algorithm is only applicable for discrete time nets.[5] Therefore, one major restriction as far as the use of this algorithm for time nets is that we cannot handle the more expressive continuous time net model.

What about discrete time nets? The clique tree algorithm is ideal for Markov time nets that have relatively few random variable for each time point. In a Markov time net, a vertex is connected only to vertices in the same, previous, and next time points. Therefore, a clique is likely to contain only vertices for those three time points, effectively capping the the size of a clique at three times the number of nodes in each time interval. The resulting clique tree ends up looking like a long narrow chain, which is efficient to process.

However, even then, on current generation workstations (Sun SparcStation-1 and -2), a practical limit is cliques of size about a dozen or so binary random variables or total clique state space of order 2^{12--16}. For discrete time nets that are not Markov time nets, the clique sizes are liable to grow much larger because edges can span multiple time points. Consequently, the clique tree algorithm appears to be less suitable for such networks.

4.4.3 Stochastic Simulation

Stochastic simulation, often known as *sampling*, is a widely used method for estimating probability distributions, including its use for estimating probabilities in Bayesian networks [Henrion, 1988a; Pearl, 1987; Shachter and Peot, 1990]. Sampling is the best available general method for estimating densities for continuous variables, and it has been employed for temporal reasoning in continuous-time Bayesian networks [Berzuini *et al.*, 1989]. A common variant of sampling, perhaps the best known, is *Monte Carlo*

[5] There are extensions allowing normal distributions, but this is not expressive enough for representing continuous time nets in general.

simulation. Monte Carlo simulation, as with sampling itself, has its origins in the physical sciences [von Neumann, 1951; Metropolis *et al.*, 1953].

The basic idea of sampling is simple. The idea is to estimate variables, in our case probability distributions, by repeated trials. In each trial, each random variable is *sampled* from its underlying distribution [Devroye, 1986; Ripley, 1987] in a fixed order. For the moment, let us pretend that there is no evidence; adding evidence requires an additional step, albeit a simple step. A sample is a simulated value for the random variable. For instance, if the random variable is the face of a tossed coin, a sample is either heads or tails. If the random variable is the date of an event, then it is some date – say, 3:43:29 p.m. tomorrow. The underlying distribution is approximately 50 - 50 for a fair coin, and for an event date, it is some parametric distribution such as a Beta distribution.

Each sample is *scored* according to some criteria. During repeated trials, a random variable takes different values with varying frequency, and the different values accumulate different scores. At the end, the scores are used to estimate the distribution for each random variable. Under certain conditions, the estimates are shown to converge to the true distribution (see, for example, [Geweke, to appear]).

In the simplest case, we design our algorithms so that the frequency with which a value is sampled for a random variable reflects the underlying distribution of the random variable faithfully, and we score each trial equally. Thus for a fair coin, we set up our algorithm to produce a coin toss of heads about half the time. Since each trial counts equally, the probability of heads is estimated at about a half, as we expect. As we will see later, we may sample and score each random variable however we like, as long as we keep track of how we do it. The basic idea is that if we produce a particular value more often than the underlying distribution warrants, then we score those trials with less weight. One benefit of such an approach is with random variables that take on some states very rarely. If a random variable takes on some state with probability one in a million, then a naive sampling method may not sample that state for a very long time! If the rare event represents, for instance, an spacecraft component failure, though, we will be interested in sampling the rare but important event as much as possible. This technique of skewing the samples is known as *importance sampling* for this very reason.

As far as the distribution to sample, it will be a marginal distribution in the case of a root node, and it will be a conditional distribution otherwise. To sample

from a conditional distribution, we must know the values of the random variables in the conditioning set of the conditional distribution. Needless to say, this applies recursively; for each variable in the conditioning set without a marginal distribution, we must know the value of *its* conditioning set. The value, by the way, may be either a sampled value or evidence as we will shortly see.

Thus, in general, a sampling algorithm of this type must sample the random variables in a sorted order, starting from the root nodes down in the direction of the edges. It is easy to show that there is always a such an order, provided that the graph is acyclic, *i.e.*, that it is a proper Bayesian network. A suitable order is given by a topological sort of the vertices on the basis of the edge relationship.

So far we have not addressed the issue of the range of the random variable. For a discrete random variable, each sample is a particular state out of a finite set. Heads or tails for a coin toss, as we saw. For this reason, scoring each sample is easy. We simply record the score in a data structure representing a set, such as a vector or list with one data cell for each alternate.

For a continuous random variable, things are not quite that simple. For instance, we cannot have a vector of all possible sample values. There are a number of data structures suitable for such use. The two simplest approaches are (1) *discretizing* the range, and (2) keeping all samples. In the first, we discretize the range of the continuous variable into a finite number of subranges, or *buckets*. In the case of continuous time nets, this corresponds to a partitioning of the time line, much as we had for discrete time nets. A bucket is scored each time a sample falls within its subrange. In the second alternative, all samples are kept, and the density for a particular range is computed on demand by inspecting all the samples. This approach is suitable if the number of samples that are expected to be needed is not large, or if we do not expect too many queries for density estimates.

In the first case, the discretization defines the resolution to which we can distinguish the occurrence of events. We must bound the range in addition to discretizing it, to limit the number of buckets. At the same time, it is sometimes convenient to create distinguished uppermost and bottommost buckets containing $+\infty$ and $-\infty$, respectively.

The two approaches are not mutually exclusive. In fact, the general approach that we take involves both a discretized density, and storing the samples, for each enabling event and clipping event of a fact. The reason for this will be seen shortly.

There is other benefit to having access to all the samples. They provide an audit trail of the algorithm's performance, and thus is valuable for empirical evaluation of its convergence statistics. Of course, a file oriented audit trail is just as suitable as an in-core one for this purpose.

Let us consider further our basic algorithm. In our basic algorithm, we score each sample equally by 1. For each node for an event, we have a discretized range, and we score a bucket at each trial. In addition, for each enabling and clipping event, we store the trial itself. At the end, we approximate the distribution of `date` nodes by examining the score in each bucket. Dividing by the number of trials gives the probability for that bucket. If we know the number of trials n beforehand, then we can score each sample by $1/n$ instead of 1. Alternatively, we can forget division altogether; the scores are proportional to the true distribution. This can be used to estimate a distribution such as `P(beg(`π`) ∈ [t, t+`ϵ`])`. Because of our discretization, our estimate corresponds to the probability that `beg(`π`)` occurred in the interval constituting the bucket that contains the time point `t`, not necessarily the probability that it occurred in the interval `[t, t+`ϵ`]`. It is also possible to compute cumulative densities simply by summing; thus it is possible to answer queries about the probability that an event takes place over a longer interval, *e.g.*, `P(beg(`π`) ∈ [u, v])?`, or even if it ever occurs at all, *e.g.*, `P(beg(`π`) < +`∞`)?` (Note that we have loosely used an event for its `date` in these example queries).

In general, we do not compute the complete probability distribution for a `range` node. Instead, the probability of a particular range is computed only on demand. To do this, we count the number of trials whose sample is the queried range. For instance, in response to a query `P([8am, 10am] =` π`)?`, we count each trial where the sample for π was `[8am,10am]`. The count is divided by the number of trials to estimate the probability. The query `P(holds(8am, 10am,` π`))?` is different from the above, because `holds(8am, 10am,` π`)` is true if π begins before 8 a.m. and ends after 10 a.m.. Thus, we would count each trial for which the sample for π contained the interval `[8am,10am]`.

The sampling algorithm has been implemented in a set of programs called `Sam`. `Sam` serves as the foundation for the temporal database developed in Chapter 5 and is described in that chapter.

With the stochastic simulation algorithm, it is simple and straightforward to reason from many types of density functions and other types of functions. All that is

necessary is that there be a function that samples or computes values from the underlying functions. There are well-known algorithms for generating standard parametric functions such as the normal density, exponential decay functions, and others [Devroye, 1986; Ripley, 1987; Rubinstein, 1981; Knuth, 1981; Press *et al.*, 1988]. It is also possible to sample the density for an event directly from its corresponding hazard function [Ripley, 1987]. The `Sam` Bayesian network evaluator implements a range of such sampling functions (see Section 5.2.6).

4.4.4 Network Processing

The performance of all Bayesian network evaluation algorithms depend on the number of nodes in the network. Therefore, if we can somehow reduce the number of nodes from the network, then the work of the evaluation algorithm will be simplified. The methods outlined here are all algorithms that may be applied for processing a network prior to using one of the evaluation algorithms given below.

We have already seen an example of how a node can be pruned out from a network. This is the example in Section 4.3.1 where we showed that the enabling and clipping events in a discrete time net can be considered superfluous, and their effects merged into the fact that they are related to.

In general, there are two conditions under which we may remove a node from a Bayesian network. One is when we are not interested in the node, and the other is when a node is irrelevant to what we are interested in. Clearly, if our interest is in computing the probability distribution for every random variable, then we cannot remove any nodes. Therefore assume that we are only interested in the distributions for some subset of the nodes V in a Bayesian network B. Can we remove all other nodes then? Well, clearly not, as we are not interested only in *output*. We are also interested in *input*, in evidence. So we cannot remove any nodes for which we might expect to receive evidence. But potentially, all other nodes may be removed. In Figure 4.19, we show an example where we eliminated all the time stages between time interval `1` and `4`.

There is a well known method [Shachter, 1986] for eliminating nodes in a Bayesian network. This is the process of *absorption*; when a node is absorbed, its children inherit its parents, and new conditional distributions are computed for each of the children. A node may not be removed under two conditions. One is when we are interested in the probability distribution at the node. In that case, it makes little.

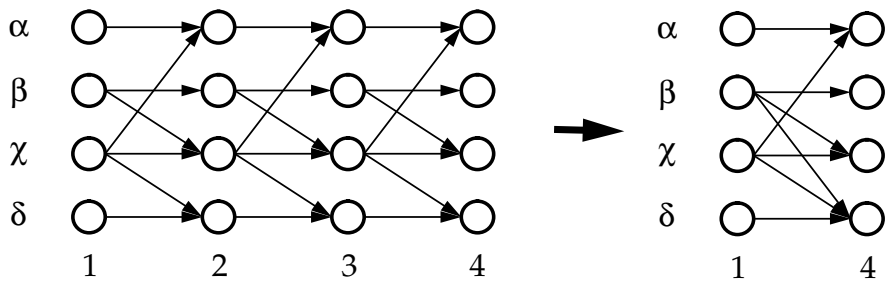


Figure 4.19: Reducing a Bayesian network

Such reduced Bayesian networks have attractive properties. Although it may still take exponential time to compute exactly with such networks, it will be of a drastically reduced order from the original network, rendering it more tractable to compute. Time complexity in the reduced Bayesian network may be further reduced if root nodes are independent of one another; then the reduced model becomes tree-like with no influences between input nodes. The time complexity of the reduction is itself exponential in the size of the set of random variables. We ran a series of experiments on discrete time nets which in their original form experienced serious combinatorial explosion in exact computation. In one set of experiments, increasing the size of a discrete time net by one time interval caused the exact computation time to increase from 15 minutes to 2 hours, and by adding another time point, from 2 hours to over 24 hours. The time for exact computation decreased by a couple of orders of magnitude when reduction was applied to the discrete time net. When we increased the number of time intervals in the discrete time net the post-reduction exact computation time increased only roughly linearly. The run-time of the reduction process itself was of the same order of magnitude as the post-reduction exact computation time.

Reduced Bayesian networks of this form are attractive for other reasons. At compile-time, reduction of the model in this form makes the Bayesian network easier to analyze and apply other compilation methods including further reduction of the model through sensitivity analysis. At run-time, the reduction of the model lead to performance improvements not only for exact algorithms, but for approximation algorithms as well.

Finally, the reduced Bayesian network is in a form that can be separated into multiple copies. In a given problem domain instance, we may have several different choices of actions to take, or several different choices of hypotheses about the world.

In such a case, we may choose to perform inference on separate copies of the Bayesian network, one per choice. Except for the overhead of the size of nodes, there is no net increase in space. With different copies for each decision, it is possible to interrupt evaluation computation with no overhead in swapping intermediate results. Therefore, depending on how well the evaluation is converging for each choice, it is possible to focus our computational resources on those choices that appear to clearly dominate others. Naturally, it is also possible to compute the expected value for each choice in parallel as well.

Unfortunately, this method is not always applicable, as there is possibility of exponential space growth in the worst case. The space tradeoff involved in Bayesian network reduction is as follows. The conditional probability matrix for a value node increases (at least doubles) for every increase of one in the size of the set of parents to the node. When we absorb a node, all of the parts of the absorbed node are added to its children's' parents. Thus, the conditional probability matrix for a node may grow fairly large as we "roll back" the nodes toward the root of the graph. Although we have not mentioned this so far, the tables do not only grow, as in some cases, we actually end up replacing two tables with one. Thus, the applicability of space reduction is highly domain specific.

4.5 Related Work

A model equivalent to the Markov time net was apparently first investigated by Tatman [Tatman, 1986; Tatman and Shachter, 1990]. Tatman developed such networks in the context of decision making using a generalization of Bayesian networks called *influence diagrams* that add decision and value variables (Chapter 6). Tatman's result was not widely known [Tatman and Shachter, 1990] at the time we developed our initial work with Markov time nets in [Kanazawa and Dean, 1989].

Henrion [Henrion,] has developed discrete-time models with continuous random variables such as our water level example in the context of air quality modeling. In this work, variables model air quality over time, given different decisions about pollution control, and there is a variable that models the value of resulting situations in terms of life expectancy of the population.

Bayesian belief networks with continuous variables for reasoning about time are due to Berzuini [Berzuini *et al.*, 1989], and perhaps to Spiegelhalter as well. Their

work grew out out of an attempt to remedy the deficiencies of the Markov time net model that we had presented in [Dean and Kanazawa, 1988b]. As it turned out, their model shares similarity with the earlier ODDS work in its application of survival analysis to temporal reasoning.

Technically, our continuous time net extends their *network of dates*. The network of dates only has what we call events. They does not incorporate a notion of facts and their probability over time, nor do they suggest how such a statistic might be recovered. So the continuous time net is a better knowledge representation for time and persistence. The continuous time net also allows arbitrary functions based on event dates and their relations, and recursively on the result of those functions. This is easily added to the network of dates but they have not done so to our knowledge.

Chapter 5

Goo

This chapter introduces Goo and family, a set of programs developed for reasoning about time and probability. The previous three chapters presented two approaches to representing knowledge about time and probability: logic and graphs. In this chapter, we consider how to combine the two in a *probabilistic temporal database*, a database for maintaining a picture of the likelihood of facts and events over time. The probabilistic temporal database incrementally constructs and maintains graph models from logical knowledge in response to queries and assertions. In this chapter, we explore the issues behind building a system combining logic-like knowledge specification and time nets.

5.1 Introduction

Goo is a system for reasoning about time and probability. Users enter facts and rules about the world, and Goo makes inferences drawing on that knowledge. It is an implementation of a combination of ideas from the previous chapters. Goo is designed first of all as a probabilistic temporal database that combines a flexible knowledge representation and query language based on the logic of lifetimes with continuous time nets for inference. It is also a knowledge-based constructor of discrete time net models and *influence diagrams* for decision analysis (see Chapter 6). Finally, its underlying Bayesian network evaluator can be used for inference in arbitrary time nets and Bayesian networks. We begin by focusing on its use as a probabilistic temporal database.

As a probabilistic temporal database, Goo supports automatic generation and augmentation of continuous time net models on the basis of knowledge expressed

in a logic of lifetimes-like language. Users supply general knowledge in the form of probability rules that link facts and events. For example, rules might give the conditional density of time of arrival in Atlantic City given the time of departure from New York and various factors such as the day of the week, whether or not it's a holiday, and so on. In each problem solving situation, users assert knowledge about actual and expected facts and events either as direct evidence or as probability densities. For example, a user might assert that a car leaves New York at 7 a.m.. Goo then combines the general rules with the asserted facts and events to create a continuous time net representing the situation. User queries about facts and events are answered by inferences based on this continuous time net. Users can query the probability of facts and events, the probability that an event precedes another event, and "what would happen if?" queries based on hypothetical facts and events.

Here is an outline of the key operations supported by Goo. Each of these capabilities are essential for any database that supports inferences about time and probability.

- Record facts and events. This involves recording the known time of occurrence of events, and when facts are known to be true. For example, for a trucking company using the database to schedule its trucks, this might involve recording the known arrival and departure times of trucks from various locations. Recording may involve adding a new fact or event to a model, or updating a model by entering evidence about an existing fact or evidence in a model. Finally, our knowledge about an event may be uncertain. In that case, we may record an event's prior density.

- Record rules. Rules are used to state the relationship between facts and events. For instance, a rule may give the conditional density for the time of arrival of a truck in Topeka given the time it left Kansas City. Other rules might state that the truck arriving in Topeka causes its location to be Topeka, or give the typical duration of a truck's stay in Topeka.

  Goo uses rules to determine which facts and events become relevant as we record new facts and events. For instance, if we record a truck's plan to move from Topeka to Cincinnati, then in order to be able to predict what time it arrives in Cincinnati, the database needs to add facts and events representing the arrival in Cincinnati and possibly various supporting facts about the route.

  Goo adds relevant facts and events automatically by applying rules. Whenever

we record a new fact or event in the database, Goo looks for applicable rules. On the basis of such rules, Goo adds facts and events, and may recursively add more on the basis of the added ones. In this manner, all a user needs to do is to assert what happens, without worrying about consequences or about forgetting relevant facts and events.

- Answer queries. An example might be as simple as 'What time is truck 20 likely to arrive in Topeka?". It may also be something more complicated like "Find me all plans that have at least 50% chance of success". Goo supports a rich class of queries including:
  - Is a fact true or false? Do we not know?
  - What is the probability that a fact is true? False?
  - What time did/will an event occur?
  - Is a fact true/will it be true at a point of time?
  - What is the probability that an event occurs in some interval?
  - What is the probability that a fact holds over some interval?
  - What is the probability that an event precedes/overlaps/..., etc., another event?
  - Negated, conjunctive, and disjunctive queries.

  Answering queries usually involves assessment of probabilities through inference.

- Hypothesize scenarios. Goo can make "What if?" inferences by recording hypothetical facts and events, and answering queries. Questions can be posed to a 'what-if' copy of the database that differs from the raw database by some hypotheses. Goo can "split" or "branch" scenarios, and to maintain a *tree* or *forest* of possible scenarios. For example, we may have a choice of where to send a truck when it finishes its current task. We may send it to Albany, or we may send it to Montreal.

Goo is a simple temporal database implemented essentially as a front end to a time net evaluator. In Goo, there is one time net that is being evaluated, updated, and modified at any given time. This time net (we will call this time net simply *the model* where unambiguous) represents the events and facts of interest to us. Goo

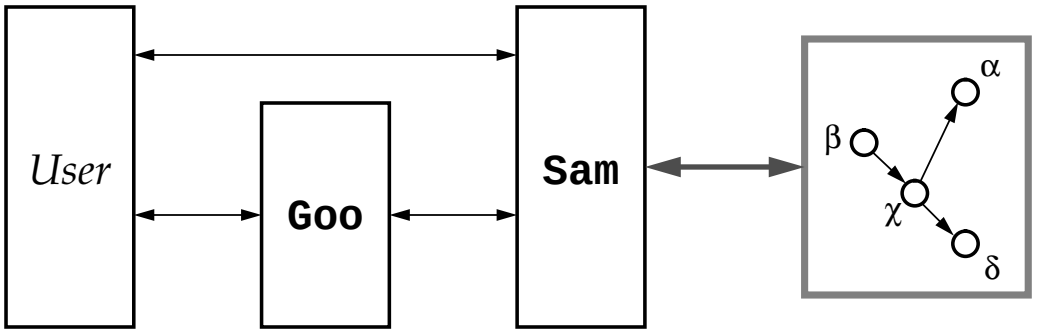


Figure 5.1: Goo architecture

incorporates an inference module, or knowledge base, that determines the model that is applicable at any given time.

In concept, the knowledge base component of Goo and its time net evaluator are independent. The knowledge base component talks to the user[1], and the time net evaluator, called Sam, assesses probabilities (Figure 5.1). The intent is that a time net is created or accessed only on the basis of database operations, for instance datum recording and queries. In practice, the interface is less restrictive as we note later.

Goo is *event driven* or *demand driven* in the sense that it does nothing without a database command:

1. User records information.

2. User queries information.

or a system command:

1. User gives [evaluator parameter, file I/O, ...] command.

In applications of Goo, the passage of time may be considered as an event as well. When Goo is issued a command, it computes until it completes the activity requested or until it is interrupted, provided that there are no other exception conditions. As a prototype, Goo does not implement sophisticated interruption or error handling facilities.

The current version of Goo is implemented in CMU Common Lisp on Sun SparcStation. Some numerical routines for random variate generation are written in C.

[1] Goo is a standalone interactive program by nature of its experiment. This does not mean that temporal databases have to be user interactive. They could be modules within a larger planning system, say, or even in a small robot controller.

This was done to avoid floating point number consing, and to take advantage of parts of an early version of `Goo` that was implemented in C and C++. `Goo` was rewritten in Common Lisp to take advantage of the more sophisticated object system for rapid prototyping of the knowledge base component.

We noted the intent of separating the functionality of the knowledge base component and the time net evaluator such that time nets are only created and accessed at a high level through the database operations of `Goo`. In practice, both are implemented as the same Common Lisp program. There is considerable separation in the functionality of the time net evaluator and the temporal database following good software engineering practice. At the same time, all evaluator commands are available to the user and interaction can proceed in a relatively free-form manner.

`Goo` maintains a simple assertional database as in deductive retrieval databases or the dynamic database in Prolog. When recording data, `Goo` first searches the database to see if any information about the assertion has been recorded. If the assertion has not been entered previously, then `Goo` enters it into the database. If it is an assertion for which there is no uncertainty, then we record its value. Otherwise, we must enter it into the current time net. If we have no current time net, then we start one. Otherwise, we augment the current time net with a random variable corresponding to the assertion. After recording the value or probability of an assertion, by adding to the time net, `Goo` propagates any effects of the change. To augment the model, we use rules representing knowledge about time and probability.

In `Goo`, we do not use the logic of lifetimes directly for knowledge base programming. Rather, we use a language that closely corresponds to the logic of lifetimes with both restrictions and extensions suitable for use as a database programming language. This language, called `Bayesl`, is implemented directly in Lisp. Knowledge about time and probability is given to `Goo` as `Bayesl` statements which are in turn just Lisp function calls. The current implementation of `Goo` supports the creation of simple theories from which to dynamically build time nets. Let us now examine in more detail the operation of `Goo` and the `Bayesl` language used for interacting with `Goo`.

5.2 Interacting with Goo

In this section, we examine Goo in more detail. We note assumptions made in its implementation, the Bayesl language, and extensions and restrictions in the knowledge base language Bayesl compared with the logic of lifetimes.

5.2.1 Propositions and Variables in Goo

The basic ontology of Goo is similar to that of the logics outlined in Chapter 3. Namely, it involves the notion of instantaneous events and persistent facts. Goo extends the ontology by introducing two kinds of variables, *simple variables* and *random variables*.

A simple variable is a logical variable that is either true or false. As such, it represents essentially a fact and it is therefore also called a *simple fact*. For all practical purposes, it is a bit that is either on or off. A simple fact is normally used to record some type of background state. An example is whether or not your spouse is en route home. Simple facts can also be used to record parameters such as parameters for probability density functions. A simple fact that has no time associated with it per se could be encoded as a fact that holds from $-\infty$ to $+\infty$. Simple facts are often binary true/false propositions. They can also be n-ary valued general propositions such as the alternate set of Chapter 4.

A random variable encodes knowledge about time and probability. As before, a random variable represents either the date of an event or the range of a fact. An example would be the time that your spouse's flight arrives. In contrast to simple facts, in its basic operation as a continuous-time probabilistic temporal database, Goo has a restriction that each proposition must be binary true/false. There is no inherent difficulty with other types of propositions. It would be possible with the addition of some extra machinery, including declaration of the types of the propositions, and their allowable states. The restriction represents the experimental prototype nature of the Goo implementation. For discrete time nets, Goo has a facility for general n-ary random variables.

For propositional theories, Goo is neutral with regard to the syntax adopted for representing each fact or event. A user can use any Lisp expression to represent a proposition including the traditional list of symbols and numbers, and strings. When the propositional content of each event or fact when represented as a list, Goo stores it

as is. Thus, by default, Goo is case-insensitive with regard to the propositional content of each event or fact: a sentence such as (holds (location Sally factory)) would normally print as (HOLDS (LOCATION SALLY FACTORY)). There is a user-controlled switch to print it in lower case instead; in general, a user may supply a function for printing propositions. It is also easy to circumvent this by representing the propositional content as, for instance, a string instead, *i.e.*, "(holds (location Sally factory))", or indeed "holds(location(Sally, factory))". Goo only requires that all references to a particular proposition be of the same type – Lisp list or string. In the case of strings, they must match exactly, including case.

Goo also handles propositional schemata that apply to all instances of a given situation. A Goo propositional schema involves the addition of variables to propositions. To simplify the interface, Goo restricts the syntax of propositions used as schemata to be a list of Lisp objects. Any Lisp symbol beginning with the question mark is regarded as a variable, *e.g.*, ?foo. We show concrete examples of propositional schemata, called *rules*, later on.

5.2.2 Dates and Times in Goo

Because Goo is a temporal database, it must represent date and time constants. A date constant represents an actual time, usually relative to some reference time. "3:00pm" is a date constant. We can think of midnight of the current day as the reference for that date constant. A time constant represents the cardinality of a time interval. "5 minutes" is a time constant.

In Bayes1, date constants are represented with an expression of the form #@"date string", where #@ is a Common Lisp 'read macro', and the double quote delimited date string names a date. Underlying Lisp routines are used for parsing the date string, allowing flexibility in the specification of dates. Examples of date constants are #@"July 11 3:23 pm" and #@"noon", the latter of which is assumed to represent noon of the current date. An optional + or – after the @ indicates "next day" and "previous day" respectively; *e.g.*, #@+"1am" means 1 a.m. tomorrow morning, and #@–"11pm" means 11 p.m. yesterday evening.

There is a special date which is associated with an event that does not occur. This date is positive infinity, and is represented as #@"infinity", #@"infty", or #@"never". Negative infinity is represented as #@"-infinity" or #@"-infty".

Apart from the infinite date, the actual value of a date expression is an integer

value denoting the elapsed time relative to a reference time (by default, the system's reference time, which is January 1, 1970) in a user-specifiable unit (by default, in minutes).

A time constant is represented by an expression of the form #!"time string", where #! is again a Lisp read macro. Again, underlying Lisp routines are used to parse the time string. Examples of time constants are #!"0:05" meaning 5 minutes, and #!"3:00" meaning 3 hours. An infinite time interval is represented as #!"infinity", #!"infty", or #!"never".

Because `Bayesl` allows the value infinity for dates and times, arithmetic and comparison with dates is performed with special versions. These are +@, -@, *@, /@, <@, <=@, >@, >=@, =@, and /=@. All are diadic operators. All the arithmetic operations return infinity if one of its arguments is infinity. Otherwise, they perform the usual arithmetic operation. The comparison relations operate analogously. A date can be checked for infiniteness with the infinity? predicate.

In specifying a `Bayesl` theory, there is an implicit *time window* for which the theory is being specified. In the logic of lifetimes, there is no restriction on the time points that a theory can refer to. An event can take place infinitely far back in the past and infinitely far into the future, and similarly for facts. In fact, many density functions can have tails with infinite extent. Practically speaking, however, it can be non-trivial to represent an infinite time line. This is especially true for `Goo` as we shall see later, because it uses stochastic sampling with a discretized time line for inference. For this reason, it is practically important to introduce minimum and maximum extents where inference will be focused on. This is similar to the eti and lti that we introduced in Chapter 4.

The lower bound of the time window is called the *time reference* or *reference time*, and the upper bound is called the *time horizon*. To designate the time reference, the set-time-reference! form is used, and for the time horizon, the set-time-horizon! form is used:

```
(set-time-reference! #@"9am")

(set-time-horizon!   #@"9pm")
```

As noted before, a user may also designate the basic unit of time. This is done with the set-time-unit! form, where the single argument designates the time unit in terms of seconds:

```
(set-time-unit! 60)
```

The above sets the base time unit to 60 seconds, or a minute, which corresponds to the system default for Goo.

5.2.3 Assertions in Goo

To assert facts and events in Goo, the record form is used. For example, to record that Sally left home at noon, we might use the following form:

```
(record (occ #@"12:00" (leave Sally home)))
```

In general, a record expression takes the form

```
(record <assertion>)
```

where <assertion> is a ground sentence of the following type:

- (true <fact>)

  Simple fact <fact> is true.

- (false <fact>)

  Simple true/false fact <fact> is false.

- (holds <fact>)

  n-ary valued simple fact <fact> is true.

- (occ <t> <event>)

  <event> occurred at time <t>. <event> may or may not be in the current model.

- (extent <t1> <t2> <fact>)

  Fact <fact> becomes true at <t1> and becomes false at <t2>.

The record form is also used to specify the alternates of an n-ary valued simple fact:

- (values <fact-value-expression>)

  <fact-value-expression> is of form (predicate { alternate1, alternate2, ... }). This states that in simple facts of form (predicate ?X), ?X can take on values from the set { alternate1, alternate2, ... }.

Finally, we also use the record form to record a rule:

- (pdf <event> <antecs> <fun>)

  The conditional density of <event> given <antecs> is represented by the function fun. <antecs> is a set (list) of random variables that <event> is directly dependent on. It may include other events, facts, and simple facts. If <antecs> is the empty set (list), then this is the prior density of <event>. <antecs> is known as the *antecedent set* or *antecedent(s)* of the rule, and <event> is known as the *consequent* of the rule.

In Goo, each event probability density function is represented with a Lisp function. Goo employs a stochastic simulation algorithm for inference. Therefore, it is convenient to let each Lisp function *sample* from the density that it represents, based on the sample values of the random variables (typically other events) that the event depends on (if any).

As an example, to represent the fact that arriving in Times Square usually occurs about half an hour after leaving home, we might use the following:

```
(pdf (arrive TimesSquare)
     ((leave home) (drive home TimesSquare))
     (lambda ()
       (norm (+ (date-of (leave home)) #!"0:30") #!"0:15")))
```

where norm is a numerical function that samples from a normal density, in this case with mean at 30 minutes after the time of leaving home and a variance of 15 minutes. The Bayesl (date-of <event>) construct is translated at run-time into an expression that returns the current sample or evidence date of the event named <event>.

Goo does not currently allow the specification of the distribution of a fact. Goo assumes that the probability density function of a fact can be recovered from the probability density functions of its enabling and clipping events. This is not the restriction that it may seem since the enabling and clipping densities can always be recovered from the distribution for a fact.

Note also that the extent assertion in Bayesl is different from a holds assertion in the logic of lifetimes. Whereas in

```
holds(12pm, 1pm, traffic-jam)
```

the traffic jam may have begun before noon and ended after one o'clock, in

extent(12pm, 1pm, traffic-jam)

the traffic jam begins at noon and ends at one o'clock. Bayesl disallows the first type of assertion because it is underconstrained.

5.2.4 Queries in Goo

A query in Goo takes one of the following forms:

- (true <fact>)

  Return whether or not simple fact <fact> is true. <fact> may contain variables if it is an n-ary valued simple fact. In that case, Goo will return the current value, if any, for that variable.

- (P <wff>)

  Return the probability of a ground sentence <wff>.

For example, (P (occ #@"noon" #@"1pm" (beg traffic-jam))) is a query for the probability that the traffic jam begins between noon and 1 o'clock.

The allowable wffs in probability queries are the following:

- (occ <t1> <t2> <event>)

  Probability of <event> occurring between <t1> and <t2>. Equivalent to

$$\int_{t1}^{t2} f_{<event>}(u)du$$

- (occ <t> <event>)

  Probability that <event> occurs at time <t>. This translates to (P (occ <t> <t+time-slice> <event>)), where time-slice is a small interval of time that is used as the granularity for time discretization by the Sam time net evaluator.

- (holds <t1> <t2> <fact>)

  Probability that <fact> holds true throughout the interval from <t1> to <t2>. If <fact> is a simple fact, then this is 1 for all <t1> and <t2>.

- (holds <t> <fact>)

  Probability that <fact> holds true at time <t>. Equivalent to (holds <t> <t> <fact>).

For queries involving a time interval [t1, t2], `Bayesl` allows t1 to be greater than t2. In other words, `Goo` regards (holds #@"1pm" #@"noon" <fact>) as equivalent to (holds #@"noon" #@"1pm" <fact>). This is reasonable for `Goo` because a date expression such as #@"noon" should denote a unique point in time; in this case "noon today".

In addition to simple queries of the above type, essentially corresponding to simple wffs, it is possible to recursively form compound queries involving relational and logical connectives:

- (<rel> <date1> <date2>)

  <rel> is a temporal relation, which can be one of <, <=, =, >=, >, or /=, the last of which is the inequality relation. <date1> and <date2> are either date constants or expressions of form (date-of <event>). Thus the relations can be used to compare two events with each other, or the time of one event in relation to a fixed point in time. Comparing two date constants is not usually very useful.

- (not <wff>)

- (and <wff1> <wff2> ...)

- (or <wff1> <wff2> ...)

- (<rel> <fact-or-event> <fact-or-event>)

  <rel> is one of the 13 primitive interval relations defined by Allen [1983], before, after, equal, meets, met-by, overlaps, overlapped-by, during, contains, starts, started-by, finishes, and finished-by. These relations are defined trivially in terms of the event and logical relations above as in Section 3.2. For example, for two facts X and Y, during(X, Y) is equivalent to:

  ```
  (and (< (date-of (beg Y)) (date-of (beg X)))
       (< (date-of (end X)) (date-of (end Y))))
  ```

  These relations are generalized to the case of events by regarding the enabling and clipping times of the event as equal to the date of the event.

Note that all the relational and compound queries involve discrete binary-valued propositions. As such, they do not correspond to continuous probability densities, but to discrete probability distributions.

Although Goo can compute probabilities for relational and compound queries, there is no way to know a priori which of such queries will be needed. Since there are an unbounded number of such queries, Goo does not represent them in its current model unless specifically asked by a user. Note that Goo may actually implicitly use such temporal or logical relations within the scope of its probability density estimates. For instance, a fact holds true for all time points greater than its enabling event date and less than its clipping event date.

Because Goo does not represent event relations explicitly in general, it cannot simply answer a compound or relational query that has not been queried already. It must explicitly add a corresponding entity in the current model and then re-estimate probabilities using the extended model. The one exception is with the not compound query, as that is simple to compute as the probability of a complementary event, with probability that sums to unity.

5.2.5 Rules in Goo

Let us now consider how knowledge about how facts and events affect other facts and events are applied in Goo. This is the province of rule application. Goo applies rules for the following purposes.

- Determine if any variables need to be added to the existing model.
- Determine the probability functions in the model.

More concretely, this means, first of all, in the case of recording data, the following.

- Determine the probability function of a newly added node.
- Determine if any other nodes should be added when adding a node.
- Determine if the probability function of any existing nodes need to be updated to reflect the changes in the model.

Secondly, as we noted before, Goo also uses rules for queries. A query may involve the derivation of relations from known entities to the requested information. For instance, a query may ask "What is the probability that `flight1437` will arrive before `flight989`?". We may know the probabilities for each separate flight, but not explicitly their relation. To compute it, we use a rule to augment the model. When

answering a query, determine what nodes or arcs need to be added to the time net, if any, and the associated probability function.

In Goo, each rule is recorded as a Bayesl pdf form as we noted before. An example of a Goo rule is the following:

```
;;; Rule for go-plan

(pdf (arrive ?agent ?dest)
     ((plan (go ?agent ?source ?dest))
      (parameters (go ?agent ?source ?dest) ?delay ?var))
     (lambda ()
       (normal (+ (date-of (leave ?agent ?source)) ?delay) ?var)))
```

This rule is an example of a propositional schema. The rule says, first of all, that it applies whenever there is a plan for agent ?agent to go from ?source to ?dest, where ?agent, ?source, and ?dest are variables. This is given by the antecedent set, which names the condition under which it applies as a pattern. The action that the rule should take is to add the node named as a consequence. It represents the time at which the agent arrives at ?dest. It is given a conditional probability density function that is the normal density with mean and variance given by the parameters assertion.

A typical plan may consist of several steps like the previous 'plan'. For instance, to get from home to the Statue of Liberty, there may be three steps, to go from home to Times Square, from Times Square to Battery Park, and from Battery Park to Liberty Island. So Goo would apply the rule three times.

A rule *applies* whenever all of the variables named in the antecedents of the rule are present in the database. When this is true, the rule antecedent is *satisfied*. Applicability of a rule is a necessary condition for performing what is specified in the rule, but it is not a sufficient condition.

We only perform the action specified in an applicable rule when:

- There is no other rule that applies better in the situation.
- The rule has not been applied already. Nodes of the type named in the consequent of the rule have already been added, and have the same density function named in the rule.

In a given situation, any number of rules may apply. Two applicable rules whose consequents are the same and whose antecedent conditions have a non-null intersection are said to *compete*.

As an example, suppose that A and B are in the database, and that the following two rules are present.

```
(pdf C
    (A)
    <fun>)

(pdf C
    (A B)
    <fun>)
```

Then the rules compete. Both rules specify that a node named C is to be added, but which probability density function is to be selected?[2]

In general, the second form corresponds to the existence of more conditioning information and is to be preferred. In Goo, the rule that applies in such situations is the rule with the single maximal antecedent set with respect to set inclusion. In other words, the rule with the largest antecedent set that matches against the current database is preferred over the others.

What if there is no single maximal set? For instance, A, B, and C may be in the database, and there may be two rules

```
(pdf D
    (A B)
    <fun>)

(pdf D
    (B C)
    <fun>)
```

(or even a third rule with antecedent set (C A)). What do we do in that case?

In general, there is no domain independent way in which we can determine the combined effects of A, B, and C. Goo cannot make any safe assumptions about rule precedence.

Therefore, in Goo we require that there be a total order with respect to antecedent set inclusion over each set of competing rules. This may not be strictly necessary, if

[2]Assuming that the probability density functions are different – if they are the same, then C *doesn't* depend on B!

there are *prototypical interactions* between facts and events. For instance, we may have three rules that indicate that in the presence of fact φ, an event ε occurs at the same time as the event α, the event β, and the event γ. In that case, the prototypical interaction may be that ε occurs at the same time as the earliest of any of α, β, and γ. Such a combination is a prototypical interaction. In some sense, the interaction behaves like a logical gate. For this reason, some well known examples of interactions are known as *noisy-OR*, *noisy-AND*, and so on [Henrion, 1988a; Pearl, 1988]. We do not implement this in Goo. However, it would be profitable to pursue it in future work.

Note that the probability density functions for competing rules must be consistent. As an example, suppose that we have a probability density function that gives the lifetime distribution for batteries, and that it gives the maximum lifetime of any battery as ten hours. In that case, adding a qualifying antecedent that the battery is a nuclear battery *cannot* make the maximum lifetime higher than ten hours. The case of nuclear batteries should have been incorporated into the first probability density function for the probability density functions to be consistent. Goo does not check for inconsistent distributions, although adding such a check can be trivial for certain cases.

The process of applying and comparing competing rules with respect to antecedent set inclusion uses unification for matching patterns which are basically ground and unground propositions.

DEFINITION 5.1
A set of propositions α is matchwise equal to a set of patterns β whenever they both contain the same number of elements, and for each pattern $\rho \in \alpha$, there is a pattern $\kappa \in \beta$ that unifies with ρ, and for each $\kappa \in \beta$, there is a pattern $\rho \in \alpha$ that unifies with β. Define the matchwise intersection of two sets of patterns to be the following. For two sets of patterns α and β, a pattern ρ is in the matchwise intersection of α and β if ρ is in α and there is a pattern that matches ρ in β.

Now, let a *rule* be the tuple $\langle C, A, \delta \rangle$, consisting of the *consequent* C, the *antecedent set* A, and the *consequent density* δ. A is a finite non-empty set of propositions, C is a single proposition, and δ is a function of time that depends on the antecedent set. Two rules *potentially compete* whenever their consequents unify and their antecedent sets have a non-empty matchwise intersection. Two rules *compete* whenever they potentially compete and they both apply.

1. Find all rules that apply. If there are none, stop.
2. Partition them into sets of competing rules.
3. Find the maximal rule in each competing set with respect to antecedent set inclusion. Call these the candidate rules.
4. If the consequent of any candidate rule is absent, then add it, according to the density named in the rule. Goto 1.
5. Otherwise, for each density stored at the consequent of any candidate rule that is different from that of the candidate rule, replace it with the density from the candidate rule.
6. Stop.

Figure 5.2: Rule application algorithm.

Let the *rule graph* $\mathcal{G}(R)$ of a set of rules R be the hypergraph formed by regarding each tuple (C, A) for each $r \in R$ as a hyperedge.

A set of rules R is *simple* if

1. Each rule $r \in R$ is ground.
2. For each set of potentially competing rules there is a total order with respect to antecedent set inclusion.
3. $\mathcal{G}(R)$ is connected.
4. $\mathcal{G}(R)$ is acyclic.

The rule application algorithm of Goo (Figure 5.2) is guaranteed to terminate for a simple rule set. The algorithm, essentially a forward chaining algorithm, is invoked whenever a previously unrecorded fact or event is asserted. Any other node that is added to a time net is a result of applying either this algorithm or the backward chaining algorithm used in answering queries described in Section 5.2.8.

It is easy to prove that the algorithm terminates.

THEOREM 5.1

The rule application algorithm terminates for a simple rule set R.

Proof. In Step 1, if no rules apply, then the algorithm terminates. If some rules apply, then steps 1 through 4 will be undertaken at most $|R|$ times. This is because the algorithm can add at most one consequent node for each pass through steps 1 through 4. Steps 1, 3, and 4 involve looking at at most $|R|$ rules. Step 2 involves looking at at most $|R|^2$ rules. Therefore, we are guaranteed to reach Step 5 if there are any rules that apply. Step 5 involves replacement of densities for at most the number of non-root vertices in the time net, which is again at most $|R|$. Therefore, the algorithm terminates.

Intuitively, $\mathcal{G}(R)$ for a simple rule set R is equivalent to the maximal Bayesian network that will be created from application of the rules. How much of this maximal Bayesian network is created depends on the simple facts and events that are asserted (recorded) to the database. Therefore, it is trivially true that the rule application algorithm, or *model construction* algorithm, will terminate.

What about rule sets that are not simple? Let us examine this in terms of the assumptions in the definition of simple rule sets.

Assumption 2 ensures that at any given time, there is an applicable rule with a maximal antecedent set. This assumption is difficult to relax without a method of "completing" the model based on the incomplete probability distributions. The prototypical interactions that we discussed above is one method of model completion. Maximum entropy methods are another [Jaynes, 1979; Cheeseman, 1983; Lippman, 1986].

Assumption 3 ensures that there is a single time net in consideration. This assumption can be relaxed trivially, at least conceptually. If the rule graph is unconnected, then the rule set is essentially disjoint, and we can extend `Goo` to handle more than a single time net model at a time. That is essentially an implementation detail.

Assumption 4 is necessary to ensure that the resulting time net will be acyclic. If the rule graph were cyclical, then we may create a cyclical network. Suppose that there are two rules $\langle\{\alpha\},\{\beta\},\delta_0\rangle$ and $\langle\{\beta\},\{\alpha\},\delta_1\rangle$, and that α is already in the network. Then the algorithm will add an edge from α to β and vice versa, creating a cycle. This assumption can be relaxed in cases when there are certain types of *asymmetric dependencies* in the rule set [Geiger and Heckerman, 1991; Fung and Shachter, 1990].

Asymmetric dependencies are best illustrated by example. Imagine that in doing my errands, I have a choice of either going to Times Square first and then Grand

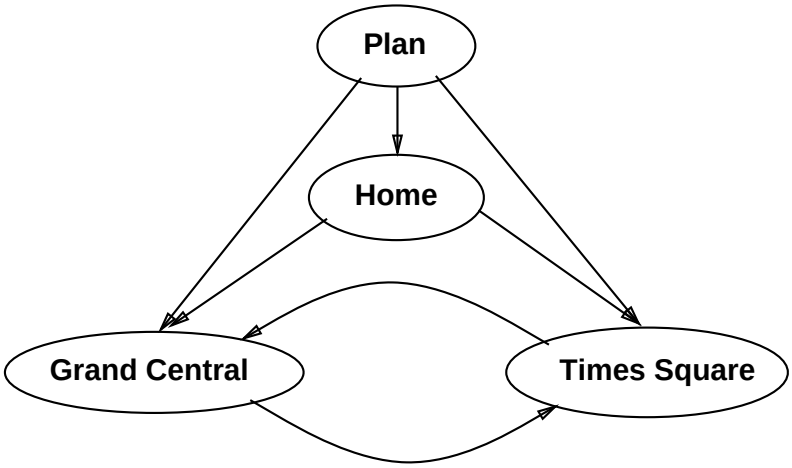


Figure 5.3: A cyclic dependency.

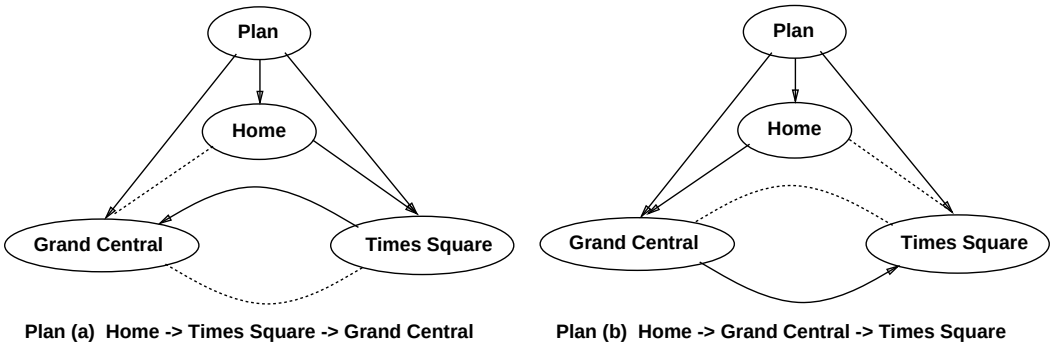


Figure 5.4: Effective dependencies.

Central Station, or Times Square first and then Grand Central. In the first case (plan a), the time of arrival at Grand Central (G) is dependent on the time of arrival at Times Square (T). In the second case (plan b), the situation is reversed. So depending on the plan (P), the arrow could point from G to T or from T to G (Figure 5.3). Thus, there is apparently a circularity in the time net representing the situation.

However, for any particular choice of plan, the time net is effectively acyclic. In Figure 5.4, we show the two alternatives with dotted lines representing the edges that are irrelevant to the plan choice. This suggests that the rule application criterion and algorithm can be revised to take advantage of this situation. Although Goo does not currently implement this, there is ongoing work in exploiting asymmetries in Bayesian networks [Geiger and Heckerman, 1991; Fung and Shachter, 1990] which will be incorporated into future versions of Goo.

Finally, assumption 1 ensures that unintended cyclicities are not created by variable substitution in unifying rules against the database. This assumption is the most difficult to relax. Goo allows users to specify rules that are not ground. However, currently Goo does not guarantee that the rule application algorithm will terminate for such rule sets corresponding to propositional schemata. It is up to the user to ensure that cycles will not be created from a rule set on the basis of a set of assertions. Note that the Bayesian network evaluator Sam will not allow the addition of edges that create cycles in a Bayesian network. Sam signals an error condition if Goo applies a rule that would result in a cycle.

5.2.6 Interacting with Sam

We now briefly introduce concepts underlying the Bayesian network evaluator Sam and its interface with Goo. Sam is a Bayesian network evaluator that uses the stochastic simulation algorithm for inference. It handles both discrete and continuous random variables, and there are various flags that can be set for selecting the sampling scheme and sampling parameters.

Sam has undergone four generations of development. The first implementation, Sam1, was a proof of concept undertaken in C. Sam2 added the capability for graphical display and construction of Bayesian networks and node distributions with X window graphics. Sam3 was a reimplementation in C++, with the addition of continuous random variables and importance sampling. The current version, Sam4, is implemented mostly in Common Lisp as noted before. Sam4 relies heavily on CLOS to implement data abstractions corresponding to graph nodes, Bayesian network graphs, and different types of algorithms and random variables. Only Sam2 features a graphical user interface, although Sam4 can generate plotting files for common plotting programs such as xgraph and GNU Plot and spawn a process to plot the file.

Interacting with Sam is as straightforward as typing to the Lisp interpreter. All external Sam functions and variables are available to Goo and its users. Sam provides a convenient syntax for referring to vertices in its time net. This is the #{node name} read macro. At run-time, #{node name} evaluates to a node object whose name is node name. Sam does not check for unique names in its vertices, so it just returns the first one it finds with the same name. A node name can be any arbitrary printing Lisp object.

Goo's primary interface to Sam consists of creating nodes, entering node evidence,

adding node parents, setting node distributions, performing inference, and retrieving distributions. In addition, there are utility functions for file I/O, parameter tuning, plotting output, and performance measurement, as well as for hypothetical inference described in Section 5.2.10. Important utility functions include those for setting the discretization for continuous time random variable nodes and for setting the sampling parameters employed in the stochastic simulation algorithm.

As noted before, `Sam` approximates continuous-time probabilistic inference using a discretization of the continuous time-line. The discretization together with the temporal window determines the granularity of the answers that `Goo` makes to its queries. The time discretization is set with the set-time-slice! form:

```
(set-time-slice! #!"0:05")
```

The above sets the time discretization granularity to 5 minutes. `Bayesl` translates this into the appropriate value given in terms of the current time unit (Section 5.2.2). The time slice value together with the time window define the range of event random variables that `Goo` creates with `Sam`.

Although there are various possibilities for sampling schemes, including importance sampling and heuristic importance sampling implemented in `Sam3`, `Sam4` currently only has straightforward likelihood weighting. The the only tuning available is the number of samples employed in simulation.

The discretization, time window, and number of samples together determine the memory requirements for each random variable. For instance, if the time reference is 9 a.m., the time horizon is 9 p.m., and the time slice is 5 minutes, then this discretization requires 12 hours $\times$ 60 minutes / 5 minutes = 144 slices. If, however, the number of samples is 1000, and the time window is much larger, then the advantages of keeping a vector of scores begins to diminish in terms of cost of memory. There is a crossover point where keeping all sample points becomes cheaper. As we noted in Chapter 4, keeping samples can be a useful and cheaper alternative depending on the problem. For facts, a full density requires n^2 space where n is the space for an event. Thus in the above example, each fact would require $144 \times 144 = 20736$, which begins to be large. In the typical problems that `Sam` has been tried on, including many different kinds of Bayesian networks other than time nets, rarely more than a few thousand samples are needed for good convergence.

5.2.7 Inference in Goo

So far, we have not said much about the actual process of inference and rule application in Goo. As before, a typical rule in Goo looks like the following:

```
(pdf (arrive TimesSquare)
     ((leave home) (drive home TimesSquare))
     (lambda ()
       (norm (+ (date-of (leave home)) #!"0:30") #!"0:15")))
```

How is such a rule actually applied?

When a rule is applicable, its antecedent set is already asserted in the database, and is present in the current time net. Furthermore, we assume that this rule has the maximal antecedent set, and is therefore the rule that Goo will actually apply. The question then is whether or not the consequent is already in the current time net, and if it is, whether or not its density is the same as the one given by the rule.

If the consequent is not in the current time net, then Goo requests Sam the addition of a new vertex corresponding to the consequent event in the rule with the density given in the rule as its conditional density.

If the consequent is in the current time net, then Goo must determine if the densities are the same. Because a density in Bayesl is represented by an arbitrary Lisp function, determining equivalence is nontrivial. Goo takes a simple approach by associating a unique identifier (number) with each rule, and noting the identifier of the source rule for the density of each time net vertex. Thus, to determine equivalence, we simply compare the identifier for the consequent density with the identifier of the rule being applied.

Recall that a probability density function is represented in Goo by a Lisp function that generates samples from the probability density function. This Lisp function takes no arguments. The probability density function form given in a rule can be of the following types:

- A Lisp symbol <fun>. In that case, this is assumed to be the name of a Lisp function that will generate samples. Sam is given (function <fun>) as the density.
- A lambda expression (lambda () ...). The expression is compiled when it is passed to Sam.
- A Lisp form (<fun> . <args>). This is assumed to be an expression that when evaluated at run-time will return one of the above.

The Lisp lambda expression and form can contain any arbitrary code. Sam provides a set of functions for sampling from standard parametric functions and hazard functions, as well as utility functions such as the indicator function. These can be used in the lambda body and Lisp form, but users can use whatever they require for the purposes of sampling.

Before the Lisp form for the probability density function can be compiled, it is preprocessed by Goo in two ways. First of all, the process of retrieving the rule should have ground the rule, replacing any variables with ground terms reflecting the current situation by unification. It is an error to try to actually apply an unground rule. Secondly, for the last two alternatives above, Goo translates each expression of form (date-of <event>) into an expression of form (date-of #{<event>}). Thus, the <event> proposition is replaced by the actual node that represents the event in the current time net. If the node does not exist in the time net, then it is an error. It is assumed that <event> is an antecedent of the rule, and therefore, that it already exists in the database. Goo checks for this at rule assertion time.

5.2.8 Answering Queries in Goo

In general, answering queries in Goo is undertaken with respect to the current time net model. If the model has not changed since the densities were last computed in the time net, then Goo will not perform inference to answer a query.

For a given fact or event, it is easy to answer a query relative to a given point or interval. Once Sam has computed densities in the current time net, each random variable corresponding to a fact or event has either a discretized density or the samples generated during stochastic simulation, or both. Then, using methods outlined in Section 4.4.3, it is easy to recover the required probability.

If the query involves a temporal or logical relation, then answering a query may involve extending the current model as noted previously. If a random variable corresponding to the query does not currently exist in the time net, Goo uses the equivalent of backward chaining rules to extend the model before performing inference and answering the query. Goo performs the model extension procedurally rather than on the basis of explicit rules, but we can regard the process as one of backward chaining rule application. We can consider each rule to have the form

(conditional-probability <relation> (<relation arguments>) <function>)

An example is the following:

```
(conditional-probability (< (date-of ?event1) (date-of ?event2))
                         (?event1 ?event2)
                         (lambda ()
                           (indicator (<@ (date-of ?event1) (date-of ?event2)))))
```

The example states that to answer a query comparing the dates of two events with the relation <, the function given as a lambda expression can be used provided that the events exist in the database. indicator implements the indicator function I: it returns 1 if its argument is non-nil, and 0 otherwise.

In general, as in the example, <relation> is an expression representing a relation, <relation arguments> are the facts or events that the relation depends on, and the <function> gives a Lisp function that returns an appropriate sample for inference.

Whenever Goo is presented with a query of type <relation>, if a node for the relation already exists, then Goo simply answers the query. If variables corresponding to the relation arguments do not exist in the database, then Goo fails and notifies users of the failure. This is essentially an "I don't know" answer. Otherwise, Goo creates a node for the relation with conditional probability given by a function such as the one in the example.

Goo implements a standard set of functions to simplify the process of generating the sample for relations. We have already described the date-of function that returns the current evidence or sample date of an event. The date of enabling and clipping events of facts can be recovered using date-of. There is also a corresponding range-of function that returns the current evidence or sample range of a fact by multiple value.

Point event relations can be handled using a scheme like the example above. For fact interval relations, Goo supplies Lisp functions that implement all 13 Allen-style relations in Lisp. These functions take as arguments nodes representing facts or events, and computes the relation based on the current evidence or sample range of the fact arguments.

To implement logical relations, Goo supplies functions holds and occ that implement the semantics of those sentence formers. The function holds is used as follows:

```
(holds <t1> <t2> <fact>)
```

where <t1> and <t2> are Bayesl time points, and <fact> is a node representing a fact. holds returns a non-nil value if the current evidence or sample range of <fact> contains (<t1>, <t2>). The function occ is implemented similarly:

(occ <t> <event>)

where <t> is a Bayesl time points, and <event> is a node representing an event. occ returns a non-nil value if the current evidence or sample value of <event> is <t>. Using holds and occ, it is trivial to implement functions that implement Bayesl logical relations as the equivalent Lisp function.

5.2.9 Updates in Goo

In the above examples, we outlined how Goo constructs a time net in response to asserted data or to requests for assessment. What if we assert data that was previously entered in the database?

If Goo finds the datum, it first checks the type of variable. A simple variable once asserted can only be a simple variable. Similarly, a random variable can only be a random variable. If there is a contradictory use, *i.e.*, a variable is declared simple at one time and uncertain at another, then Goo reports it as an error. If the datum exists in the database, and it is the right kind of variable, then we attempt a database update.

In Goo, the only database update operations allowed are for random variables, and then in the following restricted case:

- Record the (previously uncertain) date of an event.

For example, if the time of arrival of a truck in Topeka was previously unknown, we can update it by (record (occ #@"11:20am" (arrive truck Topeka))).

All other types of update operations, except those indirectly undertaken as a result of rule application, are errors. Note that this disallows the explicit retraction of previously asserted facts and events, except to erase the entire database (Erasing the database is done with the (reset-database!) form). Obviously, retraction is an important capability. For instance, we may learn that a truck never left Topeka. A more complicated example might involve learning that the truck never arrived in Topeka, but is reported to be in St. Louis instead. Updating in a general sense thus

may involve retracting some facts, and (re)recording others. For instance, we retract that the truck left Topeka, and re-record the fact that it is in Topeka.

Unfortunately, implementing retraction correctly can be non-trivial. For instance, if we are to retract the fact that the truck left Topeka, we must erase the fact that it is en route and all things that depend on its being en route. We do not want re-recording to forward chain just exactly as if it became true at that moment. This is essentially a question of proper dependency maintenance, or of bookkeeping. The simplest approach is expensive but always right: erase the database, and record all true facts again from beginning. As it happens, this is also the worst case. The issue for the bookkeeping, then, is just how much of such duplication and history keeping to avoid. Note though, that the idea may not be all that far fetched. If dependency mechanisms are very expensive, then it may simply be cheaper to redo everything. In any event, Goo does not currently implement such a capability.

5.2.10 Hypotheses in Goo

Goo implements a simple capability for performing hypothetical reasoning. By hypothetical reasoning, we mean essentially a counterfactual reasoning capability, of positing facts or events that are not known to be true, setting the time of occurrence of events to values other than their true values, and so on.

Goo allows hypothetical inference in a simple fashion by allowing the creation of copies of the database. A user can perform anything at all on a database copy, including adding new rules. This does not affect other copies in the database.

The model is stack oriented. (push-database) saves the current state. (pop-database) restores the previously saved state. (pop-all-database) unwinds the stack to the beginning. Goo allows copying the database to the limits of virtual memory. It would be easy to extend the current mechanism to push state to files.

Note that a more sophisticated hypothetical inference scheme may require the ability to retract hypothetical facts and events when restoring the database. Since Goo does not allow retraction, it cannot perform hypothetical inference except in terms of copies of the database.

5.3 An Example

```
;;; Arrival at Times Square

(pdf (arrive times-square)
     ((leave home) (drive home times-square))
     (lambda ()
       (norm (+@ (date-of (leave home)) #!"1:00") #!"0:30")))

;;; Leaving Times Square depends on the Traffic Jam

(pdf (leave times-sq)
     ((arrive times-sq) (beg traffic-jam) (end traffic-jam))
     (lambda ()
       (if (or (<@ (date-of (arrive times-square))
                   (date-of (beg traffic-jam)))
               (>@ (date-of (arrive times-square))
                   (date-of (end traffic-jam))))
           (norm (+@ (date-of (arrive times-square)) #!"0:15") #!"0:05")
           (norm (+@ (date-of (arrive times-square)) #!"0:30") #!"0:15"))))

;;; Being in Times Square

(pdf (beg (loc times-square))
     ((arrive times-square))
     (lambda ()
       (date-of (arrive times-square))))

(pdf (end (loc times-square))
     ((beg (loc times-square)) (leave times-square))
     (lambda ()
       (if (<@ (date-of (leave times-square))
               (date-of (beg (loc times-square))))
           (error "Oops, I left Times Square before I got there")
           (date-of (leave times-square)))))
```

Figure 5.5: `Goo` code for the Times Square example.

Let us now examine an example of the use of Goo for dynamic reasoning about time and probability. In this example, we are trying to determine whether it is worth our while to drive through Times Square on our way to the Statue of Liberty in New York City. We know that there are inevitably traffic jams in Times Square in the early afternoon, and that if we get caught in a traffic jam, then we will be stuck for a long time. So we want to be able to predict what time we are likely to arrive in Times Square, whether or not the traffic jam has already begun, and to predict our eventual time of arrival at the Statue of Liberty.

Figure 5.5 shows the part of the Bayesl causal theory used for reasoning about when we are likely to arrive in Times Square and how long we are likely to be there once we arrive. The rules should be self-explanatory. The first rule gives our time of arrival at Times Square as a normal distribution with mean at an hour after our time of departure from home, and a variance of 30 minutes. The time at which we leave Times Square depends on whether or not the traffic jam is going on at the time at which we arrive. Two normal densities with differing means and variances are given for the two cases. Note that this is an approximation, in that the rule only depends on whether or not the traffic jam was going on when we *arrive* at Times Square: even if the Traffic Jam begins one second after we arrive, this rule gives my time of departure from Times Square as the case where there is no traffic jam occurring. A different model might give it in terms of a proportional hazard model [Cox, 1972]. The rule set is closed with the rules that give the time of occurrence of the enabling and clipping events of our location being Times Square given the arrival event for the former, and the leaving event for the latter. These two rules have a Dirac pulse density with the pulse at the same time as the antecedent or conditioning event.

In addition to the rule set shown above, there are also similar rules for reasoning about when we will arrive at the Statue of Liberty. Given such a Bayesl causal theory, Goo constructs the appropriate model automatically on the basis of asserted facts and events and queries. Here is what such a session might look like:

```
Goo> (reset-database!)
NIL
Goo> (record (true (drive Home Times-Square)))
NIL
Goo> (record (true (drive Times-Square Statue)))
NIL
```

First of all, we flush the database with reset-database!. That erases the current contents of Goo's database, and starts a new time net. Then we record the plan to go to the Status of Liberty from Home via Times Square. In this example, we have cut out the step of going through Battery Park to keep the example manageable. We inform Goo of our plan by simple facts. At this point, Goo's time net is empty.

Next, we record the time at which we are planning to leave home, which is 11 o'clock. Goo immediately adds a node to the current time net representing that event. In addition, it immediately begins the process of forward chaining on the basis of the rules that it knows about. As it happens, it adds quite a few event and fact nodes on the basis of the plan and leave event. The nodes correspond to the entire situation of when we arrive at and leave from different locations. This is illustrated below. All the messages about compilation refer to Lisp compiling the density sampling functions at node creation time on the basis of the pdf rules.

```
Goo> (record (occ #@"11am" (leave home)))
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node   1 (LEAVE HOME)  11:00am>
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node   2 (ARRIVE TIMES-SQUARE)  NIL>
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node   3 (BEG (LOC TIMES-SQUARE))  NIL>
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node   4 (END (LOC TIMES-SQUARE))  NIL>
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node   5 (LOC TIMES-SQUARE)  NIL>
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node   6 (LEAVE TIMES-SQUARE)  NIL>
Compiling LAMBDA NIL
```

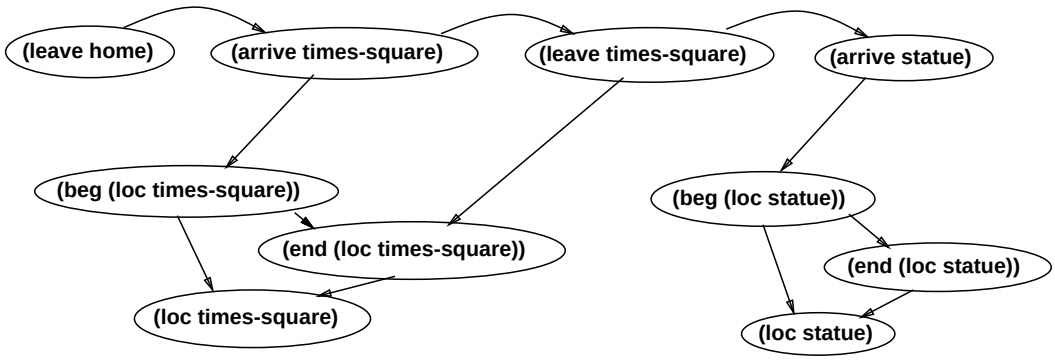


Figure 5.6: The time net after recording departure time.

```
Compiling Top-Level Form:
Goo: Adding #<Node   7 (ARRIVE STATUE)  NIL>
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node   8 (BEG (LOC STATUE))  NIL>
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node   9 (END (LOC STATUE))  NIL>
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node  10 (LOC STATUE)  NIL>
NIL
```

The time net looks like Figure 5.6. At this point, we have not added any information about the traffic jam. Let us estimate our likely time of arrival at Times Square:

```
Goo> (compute)
Goo: Computing ...
Compiling LAMBDA (#:EFFECTIVE-METHOD-GENSYM- #:EFFECTIVE-METHOD-
GENSYM- #:EFFECTIVE-METHOD-GENSYM- #:EFFECTIVE-METHOD-GENSYM-):
Compiling Top-Level Form:
NIL
Goo> (node-density #{(arrive times-square)})
#(0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.001 0.0 0.001 0.0
```

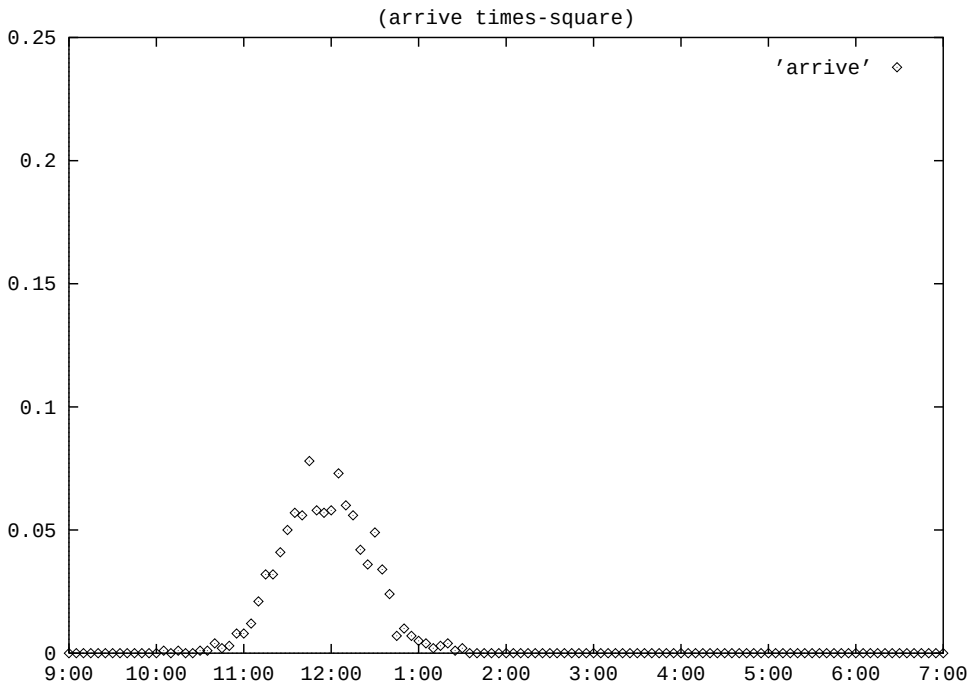


Figure 5.7: The plot for time of arrival.

```
0.0 0.001 0.001 0.004 0.002 0.003 0.008 0.008 0.012 0.021 0.032 0.032 0.041
0.05 0.057 0.056 0.078 0.058 0.057 0.058 0.073 0.06 0.056 0.042 0.036 0.049
0.034 0.024 0.007 0.01 0.007 0.005 0.004 0.002 0.003 0.004 0.001 0.002 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0)
```

The message about compilation above is due to the Common Lisp Object System compiling a class method dynamically at run-time. The density of the arrival time at Times Square is returned as a vector. As noted previously, `Sam` can output a graphics plot file of a density in some popular formats such as `xgraph` and `GNU Plot`. Here we are outputting the above vector as an `GNU Plot` format file:

```
Goo> (output-density  #{(arrive times-square)} :file "arrive.text" :type :gnuplot)
NIL
```

The result is shown in Figure 5.7.

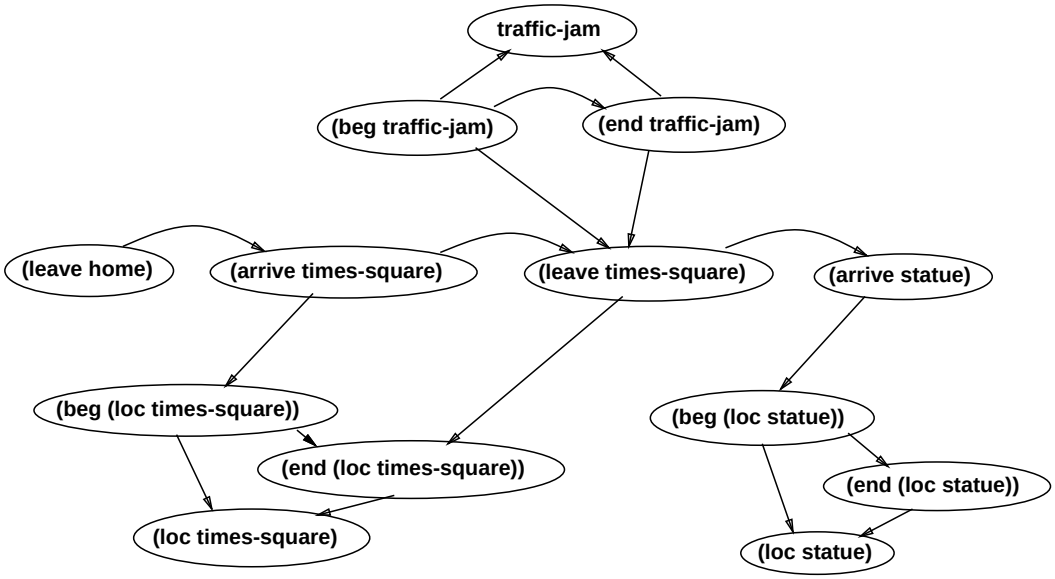


Figure 5.8: The time net after recording traffic jam.

Now, we add information about the time of the traffic jam. This results in nodes being added about the traffic jam. In addition, the conditional density for the time of leaving Times Square is replaced by a new function that depends on the traffic jam (Figure 5.8).

```
Goo> (record (pdf (beg traffic-jam) () (lambda (norm #@"12:30" #!"0:45"))))
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node  11 (BEG TRAFFIC-JAM)  NIL>
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node  12 (END TRAFFIC-JAM)  NIL>
Goo: Replacing density for #<Node   6 (LEAVE TIMES-SQUARE)  NIL>
Compiling LAMBDA NIL
Compiling Top-Level Form:
Compiling LAMBDA NIL
Compiling Top-Level Form:
Goo: Adding #<Node  13 TRAFFIC-JAM  NIL>
NIL
```

Now, let us estimate the start time of the traffic jam. When we query the node-density, Goo knows that it needs to recompute the probabilities in the time net, and goes ahead and makes the inferences. This behavior is controlled by a user settable switch.

```
Goo> (node-density #{(beg traffic-jam)})
Goo: Computing ...
#(0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.001 0.0 0.0
  0.004 0.0 0.001 0.006 0.005 0.006 0.006 0.004 0.005 0.01 0.019 0.012 0.017
  0.017 0.021 0.024 0.024 0.037 0.033 0.035 0.042 0.046 0.036 0.05 0.047 0.045
  0.039 0.04 0.044 0.035 0.033 0.037 0.035 0.031 0.031 0.018 0.019 0.013 0.021
  0.011 0.01 0.005 0.007 0.008 0.003 0.001 0.0 0.002 0.001 0.001 0.0 0.0 0.0
  0.001 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.001 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0)
Goo> (output-density #{(beg traffic-jam)} :file "jam.text" :type :gnuplot)
NIL
```

Figure 5.9 shows the plot of this density after we output it.

Let us now evaluate the likelihood that we will end up in a traffic jam if we drive through Times Square. Because this is an event relational query, and it has not been asked before, Goo must create a node representing the relational query. At this point, depending on the parameters set for Sam, Sam can estimate the distribution for this node directly, or it recomputes for the whole network again. Sam can estimate directly if it has kept around the samples generated in stochastic simulation. Because this is expensive, it is usually not turned on, as is the case here. Therefore, Goo recomputes, and returns the assessment.

```
Goo> (P (< (date-of #{(arrive times-square)}) (date-of #{(beg traffic-jam)})))
Goo: Adding #<Node  13 (< (ARRIVE TIMES-SQUARE) (BEG TRAFFIC-JAM))  NIL>
Goo: Computing ...
0.69
```

What if we now ask for the probability that we arrive in Times Square before noon? This probability can be estimated directly from the node density for the arrival event.

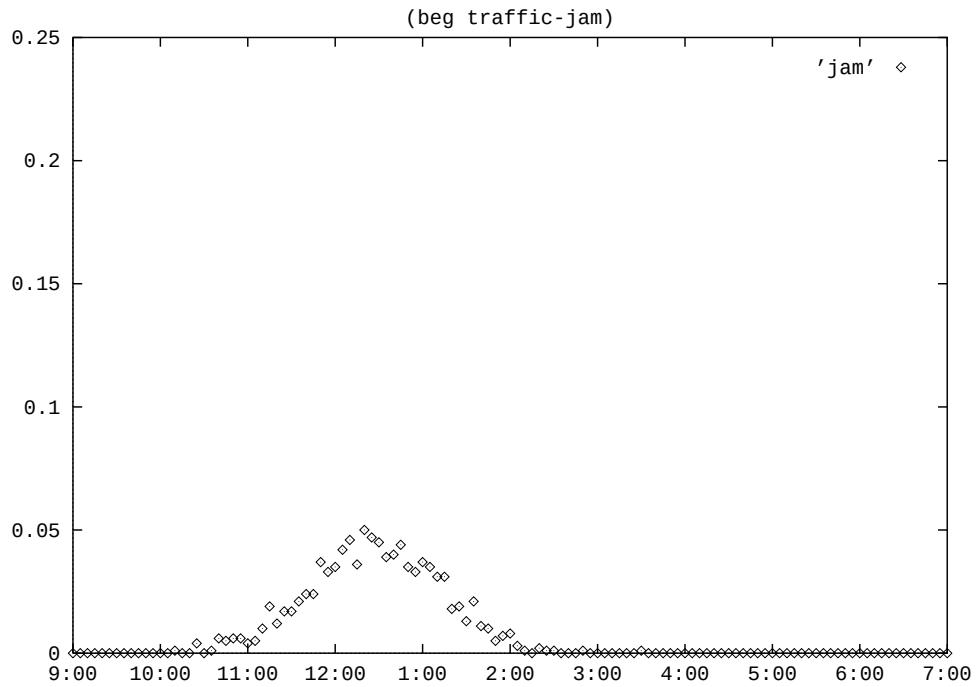


Figure 5.9: The density of the beginning of traffic jam.

For this reason, Goo will not create a node for this query, and simply returns an estimate right away without computing.

```
Goo> (P (< (date-of #{(arrive times-square)}) #@"noon"))
0.523
Goo>
```

That concludes our short example of how Goo is programmed, and how interaction with Goo can proceed.

5.4 Historical Note

McDermott considered the issues behind creating a system for managing knowledge about time in [McDermott, 1982]. Dean [1985] developed those ideas in his TMM (Time Map Manager) work. Much of the conceptual underpinnings of the current work are inspired in part by this and other related work (especially [Shoham, 1988]).

The TMM was a database for knowledge about time, facts, and events. Its representation of time was discrete, and it did not address issues of probability directly.

It did maintain knowledge about time points flexibly, however, by incorporating uncertainty in the form of temporal intervals. An event, for example, could be asserted to occur some time between two time points. A time point might itself be another event, or in reference to a global clock. The TMM always maintains constraints about facts and events, and incorporates sophisticated facilities for exception handling. Its development continues today in a more sophisticated form.

The first probabilistic temporal database was `ODDS` [Dean and Kanazawa, 1989b]. We have described the way that it handles probabilistic inference. `ODDS` implemented a very simple forward chaining deductive assertional database that incrementally built up its probabilistic model on the basis of asserted facts and probabilistic rules.

The concept of `Goo`, in its blend of logic and graphs, evolved gradually. Subsequent to `ODDS`, research in language issues were gradually put aside as sophisticated algorithms for computing probabilities were sought. In much of this research, which involved discrete time nets, the computational complexity of evaluation was such that we tended to concentrate on fixed models for each domain [Kanazawa and Dean, 1989]. However, it was always assumed that there was a reasoning system that generated the fixed models to be used for evaluation. As the research moved from discrete time nets to continuous time nets, such issues came to the fore again.

This happened because of several reasons. First of all, the continuous time net, as we noted in Chapter 4, is a flexible representation that is conceptually closer to temporal logics than is the discrete time net. The discrete time net often exhibits a more restrictive character, and makes it quite difficult to encode certain kinds of information. On the other hand, to express one fact or event, a continuous time net requires 3 nodes at most, rather than dozens or even hundreds, and the probabilities are expressed as simple functions. The continuous time net, with its ease of expressing survival models, also shares affinity with `ODDS`, raising the possibility of resurrecting the `ODDS` research agenda of creating a temporal database that handles uncertainty information explicitly with probability. As a bonus, it turned out that the natural way of encoding concepts such as events and facts had a close resemblance to the underpinnings of the temporal database work of Dean's TMM. Thus was `Goo` born; indeed, its name stands for `Grandchild of ODDS`.

In its combination of a general declarative knowledge representation with Bayesian networks, `Goo` is similar to previous work by Breese [1987], Wellman [1990], Goldman and Charniak [1990], and Horsch and Poole [1990]. With `Goo`, it is possible to build

propositional theories that specify what atoms to assert on the basis of which atoms were already asserted. The user can make requests of the form "Note this fact" or implicitly "Create this random variable as necessary". Goo tries to match that node against construction rules, and if it matches any, then it would add whatever nodes are required, and operate recursively. The relatively simple framework of Goo is perhaps most similar to Breese's ALTERID. At present, **LL** is not an object level language for describing Bayesian networks such as Poole's *Probabilistic Horn Abduction* [Poole, 1991a; Poole, 1991b], although it may be possible to make it one.

Chapter 6

Decision Making

As we have touched on, the ultimate goal of this research is to provide a way of making plans and of making decisions. We outline how the languages and models that we have developed can be applied to decision making through the use of decision theory.

6.1 Decision Theory

As we touched on in Chapter 1, the goal of this work is to develop a way of forming a picture of the world, in order to be able to know what to do. In this section, we outline how the model of reasoning about change introduced in the previous chapters support making plans and making decisions. In particular, we consider how to apply *decision theory* for the problem of planning and decision making.

Decision theory is the (mathematical) study of decision making [von Neumann and Morgenstern, 1947; Savage, 1954; Simon, 1957]. It tells us the course of action that a *rational* agent *should choose* from a set of alternatives. Decision theory in this sense is said to be a *prescriptive* theory. The opposite, a *descriptive theory*, tells us what people *do* choose. Parts of psychology constitute a descriptive theory of human decision making.

Classically, the choice that decision theory prescribes for an agent is the optimal decision, the best decision that an agent can take in a situation. The optimal decision is defined as the decision that results in the highest *expected utility*. The expected utility is the mathematical expectation, or expected value, of the utility. It is derived as the (probability) weighted sum of the utility.

The *utility* is a measure of the desirability of different situations. Since desirability is subjective, it only makes sense to talk about utility with regard to a particular subject, usually an agent, possibly a society of agents. Here, we assume that by utility, we mean utility for some single decision maker whenever we do not refer specifically to the subject.

Commonly, utility is a function that assigns a numeric value to the desirability of each different situation. Alternatively, the utility may be defined in terms of a total or partial order that indicates which situation is *preferred* over which other situations. In either case, the utility defines a *preference order* over the space of situations.

Decision theory, as it developed in the 1950s and later, was intimately linked to notions of optimal *economic behavior*, or decision making in the context of economics and business. Furthermore, some of the basic example problems concerning rational choice (as developed mathematically as well as philosophically) have to do with what a rational agent would bet in some betting situations.

DEFINITION 6.1
In the following, let our knowledge of the world be encoded in some set of random variables $X = \{x_1, x_2, \ldots, x_n\}$. *The utility is a mapping* $U : X \mapsto \boldsymbol{R}$, *where* $\boldsymbol{R}$ *is the real numbers. The expected utility of each decision* d *is simply the conditional expectation of* U *given that we take decision* $D = d$*:*

$$E[U|D = d] = \sum_{x \in \Omega_X} U(x) \times P(X = x|D = d)$$

where Ω_X *is the set of all possibilities for the random variables* X. *The optimal decision is the decision with the highest expected utility:*

$$D_{opt} = \arg\max_d E[U|D = d]$$

We defined some simple examples involving utility in previous sections. One of these involved a task τ with a deadline δ, and a *cost function cost*. The cost function is just a variation on the utility. Usually, cost is negative utility; in other words, a cost of 1 is a utility of -1. Let us recast the cost function as utility over the dates τ and δ:

$$U(\tau, \delta) = \begin{cases} +1000000 & \text{if } \tau < \delta \\ -1000000 & \text{otherwise.} \end{cases}$$

We change the example slightly and let the utility of completing task before deadline be 1000000 and the cost of not completing -1000000.

Let us briefly consider a simple example of how the expected utility would be derived. We assume that our decisions consist of $\{\alpha, \beta\}$. If we do α, there is a 0.5 probability that τ will occur before δ, and 0.5 probability that it will occur after. If we do β, then there is 0.8 probability that τ will occur before δ, and 0.2 probability of its occurring after. Note that this is a drastic simplification: in general, we have are continuous density functions giving the likelihood of occurrence of τ over the entire time line, instead of point probabilities as in the presentation here.

Let $\texttt{do}(\varphi)$ denote the act of undertaking φ. Then, the expected utility of doing α is given by

$$\begin{aligned}
E[U(\texttt{do}(\alpha))] &= E[U(\tau,\delta)|\texttt{do}(\alpha)] \\
&= \sum_{\tau} U(\tau,\delta)P(\tau,\delta|\texttt{do}(\alpha)) \\
&= U(\tau<\delta)P(\tau<\delta|\texttt{do}(\alpha)) + U(\tau\geq\delta)P(\tau\geq\delta|\texttt{do}(\alpha)) \\
&= 1000000 * 0.5 + -1000000 * 0.5 \\
&= 500000 + -500000 \\
&= 0
\end{aligned}$$

Similarly, the expected utility of doing β is given by

$$\begin{aligned}
E[U(\texttt{do}(\beta))] &= E[U(\tau,\delta)|\texttt{do}(\beta)] \\
&= \sum_{\tau} U(\tau,\delta)P(\tau,\delta|\texttt{do}(\beta)) \\
&= U(\tau<\delta)P(\tau<\delta|\texttt{do}(\beta)) + U(\tau\geq\delta)P(\tau\geq\delta|\texttt{do}(\beta)) \\
&= 1000000 * 0.8 + -1000000 * 0.2 \\
&= 800000 + -200000 \\
&= 600000
\end{aligned}$$

Since $E[U(\texttt{do}(\beta))] > E[U(\texttt{do}(\alpha))]$, the decision theoretic prescription is to do β in this situation.

The general case of the expected utility on a continuous time-line was given before. It is the integral over all costs of the product of the cost and its density:

$$E[U(\delta,\tau)] = \int_{-\infty}^{+\infty} U(\delta,\tau)f(\delta,\tau)d\tau.$$

For decision making, this integral would be conditioned on the possible actions.

Such expected utility computations form the basis of decision theory. In practice, searching for an optimal solution by maximization over the space of possible actions

is often combinatorily prohibitive. Such a search may be impractical in many situations. Simon [1957; 1975] has argued that humans often make decide on the basis of *satisficing* solutions rather than optimal solutions. A satisficing solution is a solution that is "good enough", rather than "the absolute best possible".

The idea of a decision theory based on criteria other than optimizing expected utility has seen a resurgence in recent years in various forms [Horvitz, 1987; Russell and Wefald, 1989; Boddy and Dean, 1989]. This has occurred most notably in relation to constraints about reasoning time. If there is a limited time in which to make a decision, then it may not be possible to explore all possible solutions. A decision maker in such a situation may have *bounded rationality* [Russell and Wefald, 1991], at least in the sense where rationality is defined as always doing the decision theoretically optimal action.

6.1.1 Language Issues

Can we represent decision theoretic choice by a logic-like language such as we did with probability and time? We have seen in Chapter 2 how an **LL** language can be extended to represent and reason about arbitrary quantities given by *measuring* functions. The utility function, or the cost function, is indeed a measuring function. We also discussed the addition of the expectation operator to **LL**. In that sense, **LL** already has the facility to represent decision theoretic choices. Each decision becomes, in effect, a conditioning event.

As we noted, Haddawy [Haddawy, 1991] defines a logic similar to **LL** for representing decision theoretic plans. His logic provides more facilities, and distinguishes between event occurrences and *action attempts*. Through these, he is able to provide a cleaner logic for decision theory. **LL** does not have such facilities.

6.2 The Influence Diagram

We considered the extension of Bayesian network to include utility functions in Section 4.3.2. As it happens, there is a special class of such graph models tailored for use in decision theory known as the *influence diagram* [Miller *et al.*, 1976].

The influence diagram generalizes the Bayesian network by including the effects of deterministic decisions and the utility function. An influence diagram is a directed graph $D = (N, A)$, consisting of a set of *nodes* N, and a set of *arcs* A. The nodes

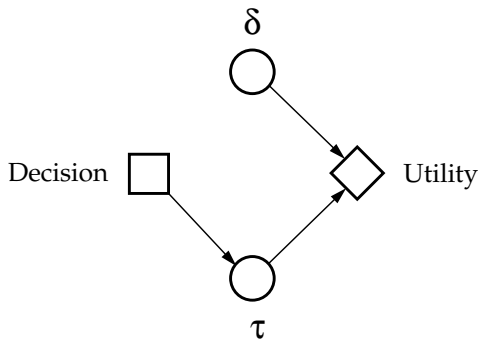


Figure 6.1: An influence diagram

consist of *chance nodes* C, *decision nodes* D, and *value nodes* V. Chance nodes represent general random variables of interest, and are essentially the same as the nodes of a Bayesian network. As with Bayesian networks, they may be discrete or continuous. Decision nodes model deterministic choices of a decision maker. They are discrete valued; each state of a decision node corresponds to a deterministic choice by a decision maker to perform a certain action. Value nodes are continuous valued and represent the utility function. A value node is essentially a special chance node. In graphically depicting an influence diagram, the following convention is often adopted: a chance node is a circle, a decision node is a square, and a value node is a diamond (see Figure 6.1).

An arc in an influence diagram is a directed graph edge, similar to an edge in a Bayesian network. Instead of representing only (probabilistic) dependence, the existence of arcs in an influence diagram indicates different things depending on the type of node that the arc points to. For each chance node c, the arcs that enter it indicate the nodes that c is dependent on. Associated with each chance node c is a conditional probability function dependent on its parents, or an unconditional (marginal) probability function if c has no parents. For each decision node d, an arc that enters it indicates *informational precedence*. If there is an arc from a node n to a decision node d, it means that at the time that the decision maker makes the decision corresponding to d, the value of the node n is known. n in this case may be a chance node or another decision node. There is no function associated with such an arc. However, it means, for instance, that a decision that precedes d must be selected before d is selected. Finally, each edge going into a value node v indicates what random variables the value node directly depends on.

In addition to the above, each arc pointing from a decision indicates what random variables (including the utility function) are directly affected by the choice of decision. A decision node must have an arc to every decision that follows it. Furthermore, in an influence diagram, there must be a total order with respect to the decision nodes. In other words, the decisions must be made in a sequence. A value node can only point to another value node, if it points to anything.

Associated with each value node is a *value function* mapping from its parents. The overall value function of an influence diagram, $\mathcal{V}$, is a function of each of the value functions of V, typically a sum or a discounted sum. In influence diagram terminology, the utility is represented by value nodes. In the standard treatment of the influence diagram, there is generally only one value node for each influence diagram. Our treatment here is completely in line with the standard treatment, however, as the set V can be aggregated into one value node as necessary. We adopt multiple value nodes as they have some representational and computational advantages. There are other constraints that define "proper" as opposed to irregular influence diagrams. For details, see [Shachter, 1986].

Thus, an influence diagram corresponds to a Bayesian network, with the addition of two things: decision nodes that may influence the state of the chance nodes and the value nodes, and a value function that evaluates the (resulting) state of the chance nodes and decision nodes. Let a *policy* be a set of unique choices for the decision nodes D. Each policy, along with any marginal distributions on the chance nodes defines a unique probability distribution over all the chance nodes. This, in turn, defines the expected utility of implementing the policy.

Shachter [1986] has developed an algorithm for computing the optimal policy (the policy with highest expected utility) on the basis of graph operations. This is the only known influence diagram specific evaluation algorithm. However, due to the similarity between the Bayesian network and the influence diagram, it is possible to use an algorithm for evaluating a Bayesian network for evaluating an influence diagram and vice versa. In our own work, we have generally applied a Bayesian network algorithm to influence diagram evaluation, as there are more efficient special algorithms for the former, as we saw in Chapter 4.

In [Kanazawa and Dean, 1989], we define a special class of influence diagram that we simply called *domain models*. A domain model is essentially an influence diagram where the chance nodes (and perhaps decision nodes) constitutes a discrete time net,

and in particular, a Markov time net.[1] In a domain model, the value function is of a special form called a *time separable value function*. By the latter, we mean the following. Let V be a set of value nodes $\{V_t : t \in T\}$, where T is assumed to be the set of time points that we are interested in. Assume that for each time point t, there exists a function U_t such that $V_t = U_t(C_{V,t}), C_{V,t} \subseteq C_t$. We say that our value function is time separable if the total value $\mathcal{V}$ is given by a function f.

$$\mathcal{V} = f(V).$$

In that case, we say that V_t is the *objective value at time t* As we noted earlier, f will be typically be a sum or discounted sum of the V_t, *i.e.*,

$$f(V) = \sum_t V_t$$

or

$$f(V) = \sum_t \gamma^t V_t,$$

where γ is a discount factor [Howard, 1971].

In the domain models of our simplified environment, a decision maker needs to make only one decision at any point in time: what plan of action to follow. Figure 6.2 depicts part of a domain model for a simple simulated autonomous "agent" [Kanazawa and Dean, 1989]. This agent is designed to model a mobile robot that moves about in a simple environment and tries to stay in operation by replenishing its energy at locations known to have energy sources. The part of the domain model in the figure encodes how certain actions, such as moving, change the robot's position with respect to locations, and models changing expectations about the energy remaining at a given location.

These domain models can be considered descendants of *Markov decision processes* [Howard, 1960; Howard, 1969; Howard, 1971]. A Markov decision process is a model in which there is a decision, state vector, and a value for each of a sequence of discrete points of time. It is equivalent to a proper Markov time net with the addition of a decision node and a value node to each time point (for a point-based Markov time net). Their similarities are the Markov assumption, the discrete temporal model, and the time separability of the value function. A key difference is that a Markov decision process represents a *sequential decision problem*, where there is a decision for each time point, whereas the domain model has only one decision. Furthermore, we noted

[1] For this reason, we have also called a domain model a *Markov influence diagram*.

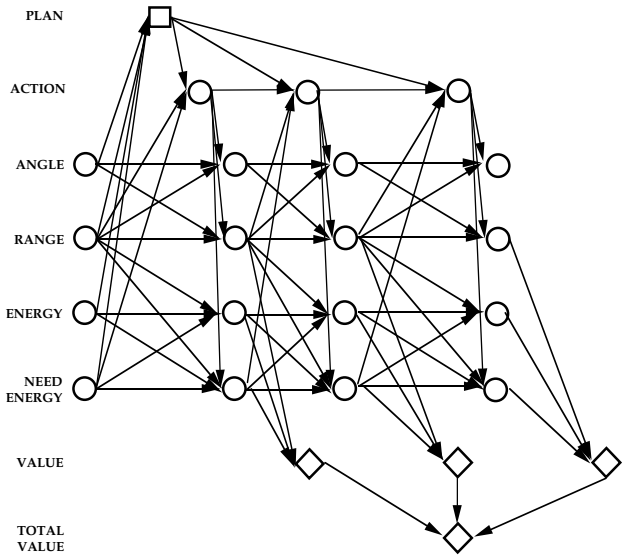


Figure 6.2: A simple domain model for a situated agent

the representational parsimony of the graph approach over the traditional transition table approach of the Markov process.

In general, the sequential decision problem undergoes severe combinatoric problems. This is because there is an exponential number of possible sequences of decisions. We have avoided some of the combinatorics of this full blown problem by adopting the plan selection decision as our sole decision. Thus, the rationale for adopting plans in our scheme is a little different from that of classical planning. We are not choosing to undertake a sequence of actions only because it is "correct" in some sense. Rather, a principal motivation is precisely its role in simplifying the sequential decision problem. Rather than facing a different decision about whether to turn right, left, eat or refuel at every possible time point, we can instead adopt a canonical sequence of actions. If we are not at a location where food exists, there is little reason to consider eating as a decision.

As it turns out, just as the Markov time net was equivalent to the Markov process, we may define a version of the influence diagram that is equivalent to the Markov decision process (see Figure 6.3). Such a model was apparently first investigated by

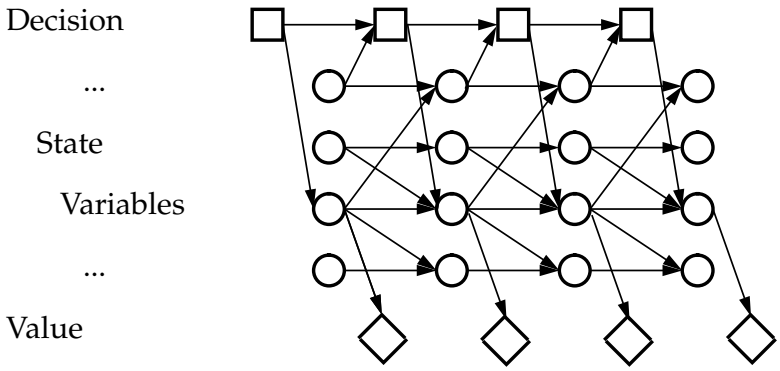


Figure 6.3: A Markov decision process influence diagram

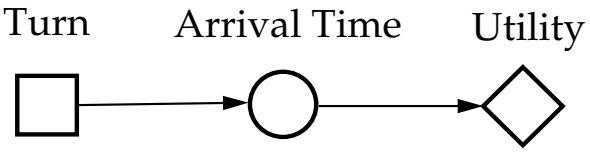


Figure 6.4: A continuous-time influence diagram

Tatman [Tatman, 1986; Tatman and Shachter, 1990]. In his paper, Tatman proves that influence diagram manipulation operations produce the same results as for the dynamic programming solution to the Markov decision process sequential decision problem. Again, such operations are of exponential order complexity.

In addition to domain models, it is also possible to define influence diagrams corresponding to continuous time nets. Again, the graph for such an influence diagram is considerably simpler in comparison with a discrete time net based domain model. In fact, the influence diagram that we depicted before (Figure 6.1) is for a continuous-time domain, representing the deadline example from Section 4.3.2. As another example, a continuous-time influence diagram might include a decision on whether to take 7th Avenue or Broadway at Times Square. Then arrival time at some location would be affected, and hence, the utility (Figure 6.4).

In the standard treatment of the influence diagram, the decision node is always a discrete-valued node. It is possible to extend this by considering a continuous-valued decision node. For instance, in controlling a mobile robot, we may wish to have a finer classification of turning beyond just right and left, to a continuous-valued motor control voltage.

As with the discrete case, in general, there is no solution for the optimization problem in the continuous case other than to consider all alternatives. Thus, finding the optimal value for a continuous decision node is like turning a dial all the way from end to end, or an antenna all the way around to find out where we get the best reception. There is no general algorithm that would, for instance, answer what the best angle is without such an exhaustive search.

In practice, we often need to discretize the parameter space of a continuous decision node first before we search for the optimal value. This is because it is impractical to "try all the floating point numbers". If the discretization is fine grain, then the state space for the decision node becomes large, and this may lead to prohibitive combinatorics. On the other hand, if the discretization is overly coarse, then the decisions may not be truly optimal. To continue the analogy of turning a dial or antenna, a discretization is akin to having notches that force the dial to jump in increments, rather than smoothly all around. If the notches are far apart (the discretization is coarse), then we may miss the frequency or antenna angle with the best reception. As we have mentioned before with regard to discretization of continuous ranges, there is no single answer that is right: a problem defines what a good discretization is.

6.3 Summary

To summarize this chapter, the framework that we have developed in this thesis is easily extended to handle decision making through decision theory. The approach developed here has been extended and applied to real mobile robots that navigate in an office environment using ultrasound and visual information [Dean *et al.*, 1990]. In addition to time net based influence diagrams such as in domain models, we have applied influence diagrams to the task of spatial inference and learning [Basye *et al.*, 1990].

As we see in the Markov decision process sequential decision problem and in the continuous optimization problem, the use of influence diagrams in temporal decision making often involves prohibitive combinatorics. We believe that this accounts at least in part for the relatively small (although growing) body of work in artificial intelligence that uses decision theory. This is an issue for decision theory in general, and has led to an active interest in models of "bounded rationality" [Simon, 1957;

Russell and Wefald, 1991]. The question of decision theoretic optimization in a continuous space is an important issue where much might be learnt from control theory [Dean and Wellman, 1991].

Chapter 7

Conclusions

We conclude by offering assessments and future directions.

7.1 Contributions

This dissertation is about how to picture the world and how it changes over time. Its chief contribution is in effecting a practical and coherent synthesis of ideas bridging representation in logic and inference in graph models for reasoning about time and probability.

Starting from the early ODDS model, we have developed a successively more expressive approach to the modeling of temporal uncertainty in the world. The primary question attacked is that of predicting the persistence of states of the world. We have also contributed in applying the framework to decision making in intelligent systems.

In this effort, we developed ways to represent time, change, and chance, including models adopted in a straightforward manner from queueing theory, graph models related to Markov process theory and survival analysis, and logics that add a probabilistic component to commonsense temporal logic. The underlying structure of knowledge and theories about time, change, and uncertainty are shared by the logic and graph models, making it easy to move from one to the other. This transparent connection promises a wider applicability of methods of temporal reasoning under uncertainty in practical problem solving applications.

We have offered algorithms for moving from general knowledge expressed in a logic-like language to graph models, and an efficient way to perform inferences for prediction in the graph models. There have been a number of implementations of the ideas

developed in this dissertation, culminating in the Goo probabilistic temporal database. Experiments with Goo bear witness to the promise of combining a general knowledge representation language with graph models for an intelligent tool for envisioning the world and for making decisions.

The models that we developed, especially the discrete-time model, have been applied successfully in real-time control of an autonomous mobile robot. We have contributed by defining the basic decision model, as well as efficient implementations for proving that it can be made to work in the real world.

7.2 Limitations

7.2.1 Logic of Lifetimes

Insofar as a language for representing facts and events and their probability, the basic **LL** languages were designed with the goal of being the simplest possible. We suggested many possible extensions in Chapter 3. A basic limitation of the current logic of lifetimes is that general atemporal random variables are not modeled. We have not offered a detailed proof theory, instead taking the expedient of performing inference in a theory by translating to time nets. It should be possible to develop an approach that performs more work in the logic rather than just in the time net. We have not defined the semantics for the first-order language. The logics do not have features for representing qualitative probability, plans, decisions, information, knowledge, introspection, desire, and belief. They are tailored for reasoning about the actions of a single agent.

7.2.2 Time Nets

Bayesian networks are still a relatively expensive computational tool. Their essential propositional character precludes the direct encoding of domain theories within the object language. We saw this difficulty surface in the different character of discrete and continuous time nets. A Bayesian network model needs to be quantitative and complete. Apart from some work in *qualitative probabilistic networks (QPN)* [Wellman, 1990], graph models generally commit the modeller to quantitatively specify all conditional probability distributions completely. Approximation algorithms to move

from quantitative to qualitative models according to error and other criteria are lacking. Ensuring consistency of the different distributions is outside the model. Finally, there are relatively few methods for learning the distributions from experience.

7.2.3 Goo

The most difficult limitation of Goo is the inability to translate arbitrary theories into time nets. It was necessary to ensure that propositional theories are acyclic to ensure termination of the model construction algorithm. Although Goo allows the specification of arbitrary theories, it does not guarantee that models will be acyclic or that the model construction algorithm will terminate. Theories for Goo need to be built with a firm understanding of the implementation rather than being truly general purpose.

Goo does not allow assertion of holds of durational facts (as opposed to simple) facts. It does not allow retraction of assertions at all, except to cycle through different alternative values for an existing proposition. Unground queries and assertions (apart from pdf rules) are prohibited. The requirement for complete rule subsumption (total order over competing rules) is restrictive.

The single model assumption needs to be relaxed in practice, but Goo cannot resolve conflicts between two different models. There is no notion of information becoming obsolete, and Goo does not know when to replace its current model by another. A user needs to know what type of time net to create a priori. Goo does not select the most appropriate type of model according to performance and error criteria. Finally, Goo only forward chains for model construction, and it is not tailored for diagnostic applications.

7.3 Future Direction

There are many directions for extending this research. These include both theoretical and experimental aspects.

7.3.1 Modeling

In terms of basic modeling for prediction, there is room for further exploration of related work in mathematics and statistics. This includes work in time series analysis,

classification and regression trees, and function approximation.

7.3.2 Languages

We have already identified many ways in which **LL** can be extended. A fruitful exercise will be to explore some of these languages, especially with regard to the representation of actions and plans, and of continuous quantities. It would be fruitful to incorporate a notion of information and observations into the languages to allow specification of informational actions and goals in plans.

Important areas for exploration include reasoning about rationality, belief, intention, and desire, especially in the context of a society of communicating agents.

7.3.3 Graph Models

The last decade has seen tremendous progress in the development of graph models. There are many remaining practical and theoretical issues in their however. From the AI viewpoint, an important limitation of these models is their basic propositional nature. It is possible that further work in the combination of flexible knowledge representation and run-time model construction offers the most promising avenue for near term gains in this regard.

Another fundamental limitation of Bayesian networks is their acyclicity. Although there are numerous proposals that relax this assumption in various ways, none are as yet adequate from the point of view of temporal and action representation. Intuitive schemes for handling cyclical dependencies are likely to lead to wider acceptance of Bayesian networks as a modeling tool within and outside artificial intelligence. Promising approaches include the identification of new types of prototypical temporal interactions that simplify specification of theories.

Another issue is not confined to graph models. This is the issue of the complexity of the real world. AI has yet to develop truly satisfactory ways of representing space, features for vision, and other things that seem essentially continuous. Graph models are not currently very good for representing these.

7.3.4 Database

Although there are many issues of interest in further development of the logic and graph models, the greatest area of obvious future work lies in further development

of the temporal database and the theory underlying dynamic management of temporal uncertainty information. One possibility is to continue the path started in `Goo`. Another is to explore the relation of the probabilistic temporal database with constraint logic programming [van Hentenryck, 1991; Jaffar and Lassez, 1987] and with non-probabilistic temporal databases such as the TMM, and combining discrete optimization methods with probabilities.

Progress on translating arbitrary theories into time nets is essential for future development. This includes automatic unfolding of cyclical theories into acyclical models. Extensions for allowing assertion of `holds` and unground queries and assertions should be relatively straightforward. A more difficult but important extension is handling multiple models with strategies for conflict resolution.

7.3.5 Action

The combination of *nonlinear planning* with decision theoretic models are a promising avenue to explore [Peot, 1992]. In planning, there is a greater need to interleave construction and evaluation of multiple models. Finally, there need to be provisions for meta-level control of inference [Good, 1976; Horvitz, 1987; Dean and Boddy, 1988; Boddy, 1991] and automatic generation of adequately accurate, as opposed to correct, models from theories based on performance criteria.

7.3.6 Learning

The `ODDS` project where it all began provided a facility for learning the probability distributions that it used for prediction from experience. It is important to incorporate learning of models into our framework, as model specification can impose a considerable burden on users. This is a wide open area, including simply learning fact duration and event occurrence statistics, learning statistics from cases, learning appropriate graph structure [Pearl and Verma, 1991; Glymour *et al.*, 1987], and induction of logical theories.

In decision-making applications, it may be equally important to bypass inference, and learn the right actions in different situations directly. Current work in this area includes work in *reinforcement learning* [Sutton, 1990; Kaelbling, 1990].

7.3.7 Applications

Last but not least, we must further explore the application of the ideas in this thesis in real applications. Domains of particular interest are medicine, distributed networks, transportation, and robotics and vision. In medicine, it is important to be able to make inferences about the likely progression of conditions in patients over time, and the effects of therapy actions over time. There is a lot of uncertainty involved in the inference and therapy planning, and the approach developed here seems ideally suited. Intelligent agents in distributed networks, or *softbots* [Etzioni and Segal, 1992], need to reason about network conditions, and the effects of actions by it and other agents over time. Because of the expected growth of the use of computer networks, such agents are promising domains for investigating planning and temporal reasoning. Transportation, both air and ground, are large scale domains in which reasoning about duration of actions is important, and because of its importance to the economy, it is another promising area. Finally, mobile robots often need to reason about the effects of their actions, what it is likely to observe over the course of time. We would like to extend our preliminary investigations in application in mobile robots into highly dynamic situations.

7.4 Coda

What have we accomplished? Did we complete what we set out to do?

The short answer to the first question is given in Chapter 1. What we have done in this thesis is to develop:

- Ways to represent our knowledge about time, change, and chance.
- Ways to efficiently extrapolate from our knowledge in order to envision the future.

In order to answer the second question, we need to have an idea of what it is that we set out to do in the first place. We informally stated our problem as that of deducing what will be true in the future, and of selecting a good action in a manner that is timely and useful.

Both problems are hard. It may be that they will never be "solved". The first problem, of prediction, will probably keep philosophers and scientists happily employed for many years to come. The second, of choice, is equally difficult. Indeed, in

our view, it is inextricably tied with the first problem; at some level, a smart choice cannot be made without some notion of what may occur. Therefore, where the first poses difficulties, so will the second.

Thus, the short answer to the second question above is a resounding no. We did not complete the task we set out for ourselves. But given the difficulty, the question in that sense is almost meaningless. We may ask instead, "Did we contribute to an understanding of the problems or make some headway in tackling them?"

The answer here must be a resounding yes. We have defined new useful models, languages for describing them, and ways to efficiently make inferences with the models. We have implemented our ideas in many different ways never before tried.

Taken piece by piece, some of what we accomplished are extensions, or turned out to be similar to what had been accomplished elsewhere. However, we have made a unique contribution in answering to both representational and computational issues. The work that we develop in Chapter 3 is similar to work in commonsense reasoning. The work developed in Chapter 4 is similar to other work in graph models. However, whereas most work in commonsense reasoning focuses almost exclusively on representational issues, most work in graph models fail to connect with ideas in commonsense reasoning. In our logics and in the time net models, we provide two languages that answer to the concerns of the two different "camps". In our case, the two types of languages share the same commitment about what to model and how to model them. In `Goo`, we began to merge the two strains, demonstrating an essential conceptual bridge toward continuing the enterprise of representing and reasoning about time and probability. In that regard, we have clearly contributed to understanding issues and in making headway in tackling them.

In the words of the great Mel Blanc,

That's All Folks!

Bibliography

[Aalen, 1989] O. O. Aalen. A linear regression model of the analysis of life times. In *Statistics in Medicine 8*, pages 907–925. Wiley, 1989.

[Allen, 1983] James Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[Allen, 1984] James Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[Bacchus *et al.*, 1989] Fahiem Bacchus, Josh Tennenberg, and Johannes A. Koomen. A non-reified temporal logic. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 2–10, Toronto, Ontario, 1989.

[Bacchus, 1991] Fahiem Bacchus. *Representating and Reasoning with Probabilistic Knowledge*. MIT Press, Cambridge, Massachusetts, 1991. Previously issued as University of Alberta Ph.D. Thesis, and as Waterloo University Technical Report CS-88-31.

[Basye *et al.*, 1990] Kenneth Basye, Moises Lejter, and Keiji Kanazawa. Reducing uncertainty in navigation and exploration. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 106–113, Cambridge, Massachusetts, July 1990. Association for Uncertainty in Artificial Intelligence.

[Berzuini *et al.*, 1989] Carlo Berzuini, Riccardo Ballazzi, and Silvana Quaglini. Temporal reasoning with probabilities. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, pages 14–21, Detroit, Michigan, August 1989.

[Berzuini, to appear] Carlo Berzuini. A probabilistic framework for temporal reasoning. *Artificial Intelligence*, to appear.

[Boddy and Dean, 1989] Mark Boddy and Thomas Dean. Solving time dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989. IJCAI.

[Boddy, 1991] Mark Boddy. Solving time-dependent problems: A decision-theoretic approach to planning in dynamic environments. Technical Report CS-91-06, Brown University Department of Computer Science, Providence, RI, May 1991.

[Box and Jenkins, 1976] George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis : Forecasting and Control.* Holden-Day, San Francisco, 1976.

[Brachman *et al.*, 1989] Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Toronto, Ontario, May 1989. Morgan Kaufmann.

[Breese, 1987] John S. Breese. Knowledge representation and inference in intelligent decision systems. Technical Report 2, Rockwell International Science Center, 1987.

[Cannings *et al.*, 1978] C. Cannings, E. A. Thompson, and M. H. Skolnick. Probability functions on complex pedigrees. *Adv. Appl. Probabil.*, 10:26–61, 1978.

[Chavez and Cooper, 1989] R. Martin Chavez and Gregory F. Cooper. An empirical evaluation of a randomized algorithm for probabilistic inference. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, pages 60–70, Detroit, Michigan, 1989.

[Cheeseman, 1983] Peter Cheeseman. A method of computing generalized bayesian probability values for expert systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsluhe, West Germany, 1983. IJCAI.

[Chow and Liu, 1968] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, May 1968.

[Cooper *et al.*, 1988] Gregory F. Cooper, Eric J. Horvitz, and David E. Heckerman. A method for temporal probabilistic reasoning. Memo KSL-88-30, Knowledge Systems Laboratory, Stanford University, 1988.

[Cooper, 1984] Gregory F. Cooper. *NESTOR: A computer-based medical diagnostic aid that integrates causal and probabilistic knowledge.* PhD thesis, Program in Medical Information Sciences, Stanford University, Stanford, November 1984.

[Cooper, 1990] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2–3):393–405, March 1990.

[Cox and Oakes, 1984] D. R. Cox and D. Oakes. *Analysis of Survival Data.* Wiley, 1984.

[Cox, 1946] Richard T. Cox. Probability, frequency and reasonable expectation. *American Journal of Physics*, 14:1–13, 1946.

[Cox, 1972] D. R. Cox. Regression models and life tables (with discussion). *Journal of the Royal Statistical Society Series B*, 34:187–220, 1972.

[Dean and Boddy, 1988] Thomas Dean and Mark Boddy. An analysis of time dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, Minneapolis, Minnesota, 1988. AAAI.

[Dean and Kanazawa, 1987] Thomas Dean and Keiji Kanazawa. Persistence and probabilistic inference. Technical Report CS-87-23, Department of Computer Science, Brown University, 1987.

[Dean and Kanazawa, 1988a] Thomas Dean and Keiji Kanazawa. Probabilistic causal reasoning. In *Proceedings of the Seventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, Edmonton, Alberta, 1988. CSCSI.

[Dean and Kanazawa, 1988b] Thomas Dean and Keiji Kanazawa. Probabilistic temporal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 524–528, Minneapolis, Minnesota, 1988. AAAI.

[Dean and Kanazawa, 1989a] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, August 1989.

[Dean and Kanazawa, 1989b] Thomas Dean and Keiji Kanazawa. Persistence and probabilistic projection. *IEEE Transactions on Systems, Man and Cybernetics*, 19(3):574–585, May/June 1989.

[Dean and Wellman, 1991] Thomas Dean and Michael Wellman. *Planning and Control*. Morgan Kaufmann, San Mateo, California, 1991.

[Dean *et al.*, 1990] Thomas Dean, Kenneth Basye, and Moises Lejter. Planning and active perception. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 271–276, Rochester, New York, 1990. DARPA.

[Dean, 1985] Thomas Dean. Temporal imagery: An approach to reasoning about time for planning and problem solving. Technical Report 433, Yale University Department of Computer Science, 1985.

[Dechter *et al.*, 1989] Rina Dechter, I. Meiri, and Judea Pearl. Temporal constraint networks. In Brachman et al. [1989], pages 83–93.

[Devroye, 1986] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, 1986.

[Etzioni and Segal, 1992] Oren Etzioni and Richard Segal. Softbots as testbeds for machine learning. In *Proceedings of the 1992 AAAI Spring Symposium on Knowledge Assimilation*, Stanford, 1992. AAAI.

[Fagin *et al.*, 1990] Ronald Fagin, Joseph Y. Halpern, and Nimrod Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1/2):78–128, July/August 1990. Academic Press.

[Feldman, 1984] Yishai Feldman. A decidable propositional probabilistic dynamic logic with explicit probabilities. *Information and Control*, 63:11–38, 1984.

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Fung and Chang, 1989] Robert Fung and Kuo-Chu Chang. Weighing and integrating evidence for stochastic simulation in bayesian networks. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, pages 112–117, Detroit, Michigan, 1989.

[Fung and Shachter, 1990] Robert M. Fung and Ross D. Shachter. Contingent influence diagrams. Submitted for publication, 1990.

[Geiger and Heckerman, 1991] Dan Geiger and David Heckerman. Advances in probabilistic reasoning. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 118–126, Anaheim, California, 1991.

[Geweke, to appear] J. Geweke. Bayesian inference in econometric models using monte carlo integration. *Econometrica*, to appear.

[Glymour *et al.*, 1987] C. Glymour, R. Scheines, P. Spirtes, and K. Kelly. *Discovering Causal Structure*. Academic Press, New York, 1987.

[Goldman, 1990] Robert Goldman. *A Probabilistic Approach to Language Understanding*. PhD thesis, Department of Computer Science, Brown University, 1990. Available as Technical Report CS-TR-90-34.

[Good, 1976] I. J. Good. *Good Thinking*. University of Minnesota Press, 1976.

[Granger and Newbold, 1986] Clive William John Granger and Paul Newbold. *Forecasting Economic Time Series*. Academic Press, Orlando, 1986.

[Haddawy, 1990] Peter Haddawy. Time, chance, and action. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 147–154, Cambridge, Massachusetts, 1990.

[Haddawy, 1991] Peter Haddawy. *Representing Plans Under Uncertainty: A Logic of Time, Chance, and Action*. PhD thesis, Department of Computer Science, University of Illinois Urbana-Champaign, 1991.

[Halpern, 1989] Joseph Y. Halpern. An analysis of first-order logics of probability. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1375–1381, Detroit, Michigan, 1989. IJCAI.

[Hanks and McDermott, 1987] Steve Hanks and Drew V. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33:379–412, 1987.

[Hanks, 1988] Steve Hanks. Representing and computing temporally scoped beliefs. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 501–505, Minneapolis, Minnesota, 1988. AAAI.

[Hanks, 1990] Steven John Hanks. *Projecting Plans for Uncertain Worlds*. PhD thesis, Yale University Department of Computer Science, January 1990.

[Haugh, 1987] Brian Haugh. Simple causal minimizations for temporal persistence and projection. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 218–223. AAAI, 1987.

[Hendrix, 1973] Gary Hendrix. Modeling simultaneous actions and continuous processes. *Artificial Intelligence*, 4:145–180, 1973.

[Henrion,] Max Henrion. Personal Communication.

[Henrion, 1988a] Max Henrion. Propagating uncertainty by logic sampling in bayes' networks. In John F. Lemmer and Laveen F. Kanal, editors, *Uncertainty in Artificial Intelligence2*, pages 149–163. North-Holland, 1988.

[Henrion, 1988b] Max Henrion. Towards efficient probabilistic diagnosis in multiply connected belief networks. In *Proceedings of the Conference on Influence Diagrams*, Berkeley, 1988.

[Horsch and Poole, 1990] Michael C. Horsch and David Poole. A dynamic approach to probabilistic inference using Bayesian networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 155–161, 1990.

[Horvitz *et al.*, 1989] Eric J. Horvitz, H. Jacques Suermondth, and Gregory F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, pages 182–193, Detroit, Michigan, 1989.

[Horvitz, 1987] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence*, pages 429–439, Seattle, Washington, 1987. Also in L. Kanal, T. Levitt, and J. Lemmer, ed., *Uncertainty in Artificial Intelligence 3,* Elsevier, 1989, pps. 301-324.

[Howard, 1960] Ron A. Howard. *Dynamic Programming and Markov Decision Processes.* MIT Press, 1960.

[Howard, 1969] Ron A. Howard. *Dynamic Probabilistic Systems*, volume I: Markov Models. Wiley, New York, 1969.

[Howard, 1971] Ron A. Howard. *Dynamic Probabilistic Systems*, volume II: Semi-Markov and Decision Processes. Wiley, New York, 1971.

[Jaffar and Lassez, 1987] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Conference on the Principles of Programming Languages*, pages 111–119, Munich, 1987.

[Jaynes, 1979] Edwin T. Jaynes. Where do we stand on maximum entropy. In Levin and Tribus, editors, *The Maximum Entropy Formalism*, pages 15–118. MIT Press, 1979.

[Jensen *et al.*, 1988] Finn V. Jensen, Kristian G. Olesen, and Stig K. Andersen. An algebra of bayesian belief universes for knowledge based systems. R- 88-25, Institute for Electronic Systems, Aalborg University, Aalborg, Denmark, July 1988.

[Jensen *et al.*, 1990] Finn V. Jensen, Steffen L. Lauritzen, and Kristian G. Olesen. Bayesian updating in recursive graphical models by local computations. *Computational Statistics Quarterly*, 5(4):269–282, 1990.

[Jensen, 1988] Finn V. Jensen. Junction trees and decomposable hypergraphs. Research report, JUDEX, Aalborg, 1988.

[Jensen, 1990] Finn V. Jensen. Personal communication, 1990.

[Kaelbling, 1990] Leslie Pack Kaelbling. *Learning in Embedded Systems*. PhD thesis, Stanford University, Stanford, 1990.

[Kahn and Gorry, 1977] Kenneth Kahn and G. Anthony Gorry. Mechanizing temporal knowledge. *Artificial Intelligence*, 9:87–108, 1977.

[Kanazawa and Dean, 1989] Keiji Kanazawa and Thomas Dean. A model for projection and action. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989. IJCAI.

[Kanazawa, 1991] Keiji Kanazawa. A logic and time nets for probabilistic inference. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Anaheim, California, 1991. AAAI.

[Kim and Pearl, 1983] Jin H. Kim and Judea Pearl. A computational model for causal and diagnostic reasoning in inference systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsluhe, West Germany, 1983. IJCAI.

[Knuth, 1981] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 2nd edition, 1981.

[Kyburg, 1983] Henry E. Kyburg, Jr. The reference class. *Philosophy of Science*, 50:374–397, 1983.

[Lauritzen and Spiegelhalter, 1988] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society Series B*, 50(2):157–194, 1988.

[Lauritzen *et al.*, 1984] S. L. Lauritzen, T. P. Speed, and K. Vijayan. Decomposable graphs and hypergraphs. *Journal Australian Mathematical Society*, A(36):12–29, 1984.

[Lauritzen *et al.*, 1988] Steffen L. Lauritzen, A.P. Dawid, B.N. Larsen, and H.-G. Leimer. Independence properties of directed markov fields. Technical Report R 88-32, Institute for Electronic Systems, Department of Mathematics and Computer Science, University of Aalborg, 1988.

[Lifschitz, 1987] Vladimir Lifschitz. Formal theories of action: Preliminary report. In Frank M. Brown, editor, *The Frame Problem in Artificial Intelligence: proceedings of the 1987 Workshop*. Morgan Kaufmann, Lawrence, Kansas, April 1987.

[Lippman, 1986] Alan F. Lippman. *A Maximum Entropy Method for Expert System Construction*. PhD thesis, Brown University, 1986.

[Martin and Allen, 1991] Nathaniel Martin and James Allen. A language for planning with statistics. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 220–227, Anaheim, California, 1991.

[McCarthy and Hayes, 1969] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4, 1969.

[McCarthy, 1986] John McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[McDermott and Doyle, 1980] Drew V. McDermott and Jon Doyle. Non-monotonic logic I. *Artificial Intelligence*, 13:41–72, 1980.

[McDermott, 1982] Drew V. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, 1982.

[Metropolis *et al.*, 1953] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953.

[Miller *et al.*, 1976] A.C. Miller, M. M. Merkhofer, R. A. Howard, J. E. Matheson, and T. R. Rice. Development of automated aids for decision analysis. Technical report, Stanford Research Institute, Menlo Park, 1976. cited by Tatman and Shachter.

[Neapolitan, 1990] Richard E. Neapolitan. *Probabilistic Reasoning in Expert Systems, Theory and Algorithms*. Wiley, 1990.

[Nilsson, 1986] Nils Nilsson. Probabilistic logic. *Artificial Intelligence*, 28:71–88, 1986.

[Nunez, 1989] Linda Mensinger Nunez. The relationship between temporal bayes networks and markov random process tables. Master's thesis, Department of Computer Science, Brown University, 1989.

[Olesen *et al.*, 1989] Kristian G. Olesen, Uffe Kjaerluff, Frank Jensen, Finn V. Jensen, Bjorn Falck, Steen Andreassen, and StigK. Andersen. A munin network for the median nerve – a case study on loops. *Applied Artificial Intelligence*, 1989. Special issue: Towards Causal AI Models in Practice.

[Pearl and Verma, 1991] Judea Pearl and Thomas Verma. A theory of inferred causation. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pages 441–452, Cambridge, Massachusetts, 1991.

[Pearl *et al.*, 1989] Judea Pearl, Dan Geiger, and Thomas Verma. Conditional independence and its representations. *Kybernetika*, 25:33–44, 1989.

[Pearl, 1986] Judea Pearl. A constraint propagation approach to probabilistic reasoning. In Laveen N. Kanal and John F. Lemmer, editors, *Uncertainty in Artificial Intelligence*. North-Holland, 1986.

[Pearl, 1987] Judea Pearl. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, 32:245–257, 1987.

[Pearl, 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, Los Altos, 1988.

[Peot and Shachter, 1991] Mark Peot and Ross Shachter. Fusion and propagation with multiple observations in belief networks. *Artificial Intelligence*, 48(3), 1991.

[Peot, 1992] Mark Peot. Conditional nonlinear planning. In *Proceedings of the First International Conference on AI Planning Systems*, College Park, Maryland, 1992.

[Poole, 1991a] David Poole. Representing Bayesian networks within probabilistic Horn abduction. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 271–278, Anaheim, California, 1991. Association for Uncertainty in Artificial Intelligence, Morgan Kaufmann.

[Poole, 1991b] David Poole. Representing diagnostic knowledge for probabilistic Horn abduction. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 1129–1135, Sydney, Australia, 1991. IJCAI.

[Press *et al.*, 1988] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, 1988.

[Ripley, 1987] Brian D. Ripley. *Stochastic Simulation*. Wiley, 1987.

[Rit, 1986] J.-F. Rit. Propagating temporal constraints for scheduling. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 383–388, Philadelphia, Pennsylvania, 1986. AAAI.

[Rose *et al.*, 1976] Donald J. Rose, R. Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal of Computing*, 3(2):266–283, June 1976.

[Rubinstein, 1981] Reuven Y. Rubinstein. *Simulation and the Monte Carlo Method*. Wiley, 1981.

[Russell and Wefald, 1989] Stuart J. Russell and Eric H. Wefald. Principles of metareasoning. In Brachman et al. [1989].

[Russell and Wefald, 1991] Stuart J. Russell and Eric H. Wefald. *Do The Right Thing.* MIT Press, Cambridge, 1991.

[Savage, 1954] Leonard J. Savage. *The Foundations of Statistics.* Dover, 1954.

[Schubert, 1989] Lenhart Schubert. Solving the original frame problem without frame axioms or nonmonotonicity or frame axioms. In Henry Kyburg and Ron Loui, editors, *Selected Papers from the 1988 Society for Exact Philosophy Conference*, 1989.

[Shachter and Peot, 1989] Ross D. Shachter and Mark A. Peot. Evidential reasoning using likelihood weighting. Technical report, Artificial Intelligence, Engineering-Economic Systems Department, Stanford University, 1989.

[Shachter and Peot, 1990] Ross D. Shachter and Mark A. Peot. Simulation approaches to general probabilistic inference on belief networks. In John F. Lemmer and Laveen F. Kanal, editors, *Uncertainty in Artificial Intelligence5.* North-Holland, 1990.

[Shachter, 1986] Ross D. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6):871–882, November/December 1986.

[Shoham, 1988] Yoav Shoham. *Reasoning About Change: Time and Causation from the Standpoint of Artificial Intelligence.* MIT Press, Cambridge, Massachusetts, 1988.

[Shwe and Cooper, 1990] Michael Shwe and Gregory Cooper. An empirical analysis of likelihood-weighting simulation on a large, multiply-connected belief network. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 498–508, Cambridge, Massachusetts, 1990.

[Simon and Kadane, 1975] Herbert A. Simon and Joseph B. Kadane. Optimal problem-solving search: All-or-none solutions. *Artificial Intelligence*, 6:235–247, 1975.

[Simon, 1957] Herbert Simon. *Models of Man: Social and Rational; Mathematical Essays on Rational Human Behavior in Society Setting.* Wiley, New York, 1957.

[Sutton, 1990] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings 7th International Conference on Machine Learning*, 1990.

[Syski, 1979] Ryszard Syski. *Random Processes*. Marcel Dekker, New York, 1979.

[Tarjan and Yannakakis, 1984] Robert E. Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal of Computing*, 13(3):566–579, August 1984.

[Tatman and Shachter, 1990] Joseph A. Tatman and Ross D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):265–379, March/April 1990.

[Tatman, 1986] Joseph A. Tatman. *Decision processes in influence diagrams: Formulation and analysis*. PhD thesis, Stanford University Department of Engineering / Economic Systems, Stanford, California, 1986.

[van Fraassen, 1980] B.C. van Fraassen. A temporal framework for conditionals and chance. In William L. Harper, Robert Stalnaker, and Glenn Pearce, editors, *IFs*, pages 323–340. D. Reidel, Dordrecht, 1980.

[van Hentenryck, 1991] Pascal van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, Massachusetts, 1991.

[Vere, 1983] Steven Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5:246–267, 1983.

[Vilain, 1982] Marc Vilain. A system for reasoning about time. In *Proceedings of the National Conference on Artificial Intelligence*. AAAI, 1982.

[von Neumann and Morgenstern, 1947] John von Neumann and O. Morgenstern. *Theory of games and economic behavior*. Princeton University Press, Princeton, 2nd edition, 1947.

[von Neumann, 1951] John von Neumann. Various techniques used in connection with random digits. *U.S. National Bureau of Standards Applied Math Series*, 12:36–38, 1951.

[Weber, 1989a] Jay Weber. *Principles and Algorithms for Causal Reasoning with Uncertainty.* PhD thesis, University of Rochester Department of Computer Science, May 1989. Technical Report 287.

[Weber, 1989b] Jay Weber. Representing and computing temporally scoped beliefs. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989. IJCAI.

[Wellman, 1990] Michael P. Wellman. *Formulation of Tradeoffs in Planning Under Uncertainty.* Pitman, London, 1990.

[Wright, 1921] Sewall Wright. Correlation and causation. *Journal of Agricultural Research*, 20:557–85, 1921.

[Yannakakis, 1981] Mihalis Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal of Computing*, 2(1):77–79, March 1981.

Index

Author Index