Prediction as a Knowledge Representation Problem: A Case Study in Model Design

P@trik Haslum
pahas@ida.liu.se

April 2002

Abstract

The WITAS project aims to develop technologies to enable an Unmanned Airial Vehicle (UAV) to operate autonomously and intelligently, in applications such as traffic surveillance and remote photogrammetry. Many of the necessary control and reasoning tasks, *e.g.* state estimation, reidentification, planning and diagnosis, involve prediction as an important component. Prediction relies on models, and such models can take a variety of forms. Model design involves many choices with many alternatives for each choice, and each alternative carries advantages and disadvantages that may be far from obvious. In spite of this, and of the important role of prediction in so many areas, the problem of predictive model design is rarely studied on its own.

In this thesis, we examine a range of applications involving prediction and try to extract a set of choices and alternatives for model design. As a case study, we then develop, evaluate and compare two different model designs for a specific prediction problem encountered in the WITAS UAV project. The problem is to predict the movements of a vehicle travelling in a traffic network. The main difficulty is that uncertainty in predictions is very high, due to two factors: predictions have to be made on a relatively large time scale, and we have very little information about the specific vehicle in question. To counter uncertainty, as much use as possible must be made of knowledge about traffic in general, which puts emphasis on the knowledge representation aspect of the predictive model design.

The two model designs we develop differ mainly in how they represent uncertainty: the first uses a coarse, schema-based representation of likelihood, while the second, a Markov model, uses probability. Preliminary experiments indicate that the second design has better computational properties, but also some drawbacks: model construction is data intensive and the resulting models are somewhat opaque.

Acknowledgements

June 15th, 1999, I sent a short paper outlining some ideas about prediction and planning in the presence of independent agents to my advisor, Patrick Doherty. If he had not taken such an interest in those ideas, or had not had the patience to wait three years for them to assume their present form, this thesis would never have been. Thanks Patrick.

As a novice researcher, I owe much to the environment in which I've had the pleasure of working, and to the people who create it: Many of the ideas presented in this thesis, not only the problem treated in the case study, have grown out of experiences gained in the WITAS project. Although it's sometimes a lot of work, it's also interesting and exciting, and I'd like to thank all members of the project team, past and present, for making it so. Through the ECSEL graduate school I've been introduced to people and ways of thought I would never otherwise have met, and this has certainly left its mark on the thesis. My thanks to those who thought it up, made it work and took part.

Several people have also provided me with very concrete help in the writing of this thesis: Linda Wahlman at Norrköpings Komun provided the traffic data used in experiments and answered a lot of questions. Marcin Szczuka provided pointers to work on learning DFAs. Marcus Bjäreland and Björn Wingman constructively criticized various drafts of the thesis. Lillemor Wallgren has been very helpful in guiding me through the administrative process of a thesis defence, in spite of my always being late.

All faults are, of course, my own.

Contents

Chapter 1

Introduction

Prediction is the process of determining the possible future developments of a dynamic system from information about the past and present observations of the system, and assigning to each possible development some measure of relative likelihood. In less technical terms, it means making guesses about what will happen[1].

The prototypical example of prediction may well be weather forecasts, but statistical prediction (also known as "forecasting") plays an important role in engineering, business, *e.g.* forcasting market demands for a product to avoid shortage and overproduction, and medicine, *e.g.* for assessing risks. What the examples illustrate is the fact that the value of prediction lies in that it helps us make more informed decisions: What to pack? Quit smoking or eat less fat? Hire or fire? And so on.

Predictive capability is equally important to an artificial autonomous intelligent agent as it is to us. In many of the reasoning tasks studied in automatic control and artificial intelligence, *e.g.* state estimation or planning, prediction is an essential component. In spite of this, or perhaps precisely because of this, the problem of prediction is rarely studied on its own.

1.1 Models for Prediction

From knowledge of the past and present alone, the future can, of course, not be known (short of appealing to the crystal ball). Prediction relies on models, that are more or less coarse descriptions of how the system is expected to evolve. The model may be derived from the physical laws of the system dynamics, from statistical measures of correlation, or simple heuristic estimates. An important point is that the model is, almost always, a simplified description of reality, which introduces into

[1]The counterfactual question, "what would have happened, had things been different?" is for all practical purposes indistinguishable from prediction.

any prediction an element of uncertainty[2].

Examining the diverse set of tasks that employ prediction reveals a variety of different ways to construct, represent and perform prediction with a model. We shall try to make a distinction between the "model design", *i.e.* the ontology, representation and the computational methods with which the model is constructed and applied, and the model as an instance of a design. Each model design carries advantages and disadvantages, for example the complexity of computing predictions or the amount and form of data needed to construct a model instance.

Because the value of prediction lies in its ability to inform and improve decisions made in the context of another task, which in any particular case depends as much on the actual model instance as the design, evaluating the model design separate from its instantiation is a difficult problem. Nevertheless, "the result can never be better than the design permits": the model design limits as well as supports different ways of construction, maintenance and use of model instances to a varying degree.

1.2 Agenda

The motivation for this inquiry rests on the following "theses". Although not particularly controversial, we feel that it is important to state them explicitly.

1. *Prediction is an essential component in many capabilities required by autonomous intelligent agents.*
2. *For any instance of the prediction problem, there are many choices and alternatives in the design of the predictive model. Each carries certain advantages and disadvantages, and evaluating model designs is non-trivial.*

For a particular task involving prediction of a particular system, this leads to the question: what model designs are there to choose from, and which should be chosen? The aim of our inquiry is, in short, to investigate the space of predictive model designs and to assess their suitability for particular tasks and systems.

That such a systematic study has not already been done[3] may seem surprising, considering the prevalence of prediction in so many important and well studied control and reasoning tasks. Because, however, prediction is useful only in context, research on each of these tasks has been focused on solutions to the whole problem. Perhaps therefore, the form of the predictive model is often taken as granted, by the nature

[2]In case the model is a perfect, or perfect enough, depiction of the system, the problem is commonly regarded as one of simulation. This does not imply that simulation is trivial, but difficulties (in modelling and in computation) are more due to the fact that systems of interest tend to be very large.

[3]To the best of our knowledge, it hasn't. Rosen (Rosen 1985) emphasizes the importance of predictive models in the study of complex dynamical systems, and to some extent discusses the characteristics of different model designs, but from a less practical perspective than what we are interested in.

of the task or by tradition: for example, state estimation typically uses models with random (Gaussian) noise, while planning uses (deteministic or non-deterministic) state/transition models.

A complete systematic study of predictive model designs is of course a gargantuan task. This thesis offers only a small beginning, consisting of two parts: The first is a survey of uses of prediction, and of the model designs used, while the second is a case study, examining in more depth two different solutions to a particular prediction problem.

1.2.1 Part I: A Survey

The "space of predictive model designs" is largely uncharted territory, in the sense that there are no frameworks or systems of classification that relate different model designs[4]. Therefore, the first part of our inquiry is a survey, examining a number of control and reasoning tasks and identifying the predictive mechanism in each.

From the examples found in the survey, we extract a number of choices in model design, examine alternatives, and issues raised by each choice. The aim is to identify a few dimensions in the design space. For a particular predictive problem, considering combinations of different positions in these dimensions may suggest alternative model designs.

1.2.2 Part II: A Case Study

Although the survey and analysis gives a tentative map of the possible designs for a predictive model, the question of suitability can not be adressed so generally. For this, our method will be case studies.

The case study, as a research methodology, is difficult to characterize, and often confused with other methodological distinctions such as the choice of qualitative versus quantitative descriptions or the exploratory, descriptive or explanatory nature of the study. Case studies are often described as "contextual", meaning that the studied phenomenon is not isolated from its context, and "holistic", meaning that many aspects of the person, group, incident or whatever entity plays the part of "case" are considered, in depth.

From the flora of attempts to define case studies, we seize upon the following: "The case study inquiry copes with the [...] situation in which there are many more variables of interest than data points." (Yin 1994). A case, for us, is the predictive component of a specific task in a specific environment. The design choices and the criteria for evaluting a model design are our "many variables of interest". Because realizing and evaluating even one, let alone several, designs is a substantial piece of work, data points are necessarily few. The results of the study of one such case

[4]As opposed to the situation in, for example, software design or experiment design, for which frameworks exist and, at least in the case of the latter, the strengths, weaknesses and range of applicability of the different methods have been studied.

should of course not be casually generalized to other tasks or other environments. Conclusions drawn from a single case study are at best hypotheses, that studies of further cases may refute.

There is, however, also a more practical point to consider: creating a predictive model, or improving on existing model designs, even if only for a specific task or system, may be a relevant problem in itself. The case study carried out in the second part of this thesis has resulted in the design, implementation and tentative evaluation of two different predictive models, at least one of which was needed anyway.

1.3 The WITAS UAV Project

The motivation for this inquiry into the nature of predictive models comes, framework-building ambitions aside, from problems encountered in the WITAS UAV project.

The WITAS project aims to develop architechtures and techniques for an autonomous Unmanned Airial Vehicle (UAV) for traffic surveillance[5]. Because of the complexity of the application, the project involves groups in diverse areas of expertise such as artificial intelligence, computer vision and automatic control.

An actual UAV ready for commercial use is not likely to be developed within the projects time frame, nor is this the highest priority. Basic research, applicable also to autonomous systems in other environments, is an important part of the project. Traffic surveillance is a suitable test domain because it is complex and dynamic enough to be challenging for vision and reasoning systems, while having enough structure not to be an impossible task.

1.3.1 Predicting Vehicle Movements

The planning of a search strategy and the reidentification of a previously observed vehicle are two examples of tasks that can benefit from prediction of vehicle movements.

In both cases, predictions have to be made based on scant information and on a relatively large time scale. A predictive model can therefore not rely on vehicle dynamics alone: the behaviour and intentions of the driver, and the surrounding traffic situation, plays a large part in determining where it is going to be.

As a consequence, the representation and use of knowledge will be important in the two model designs we develop. The main point of difference between them is how the relative likelihood of different future developements is determined: in the first, it is a "qualitative" measure based on a heuristic (or "common sense") model, while in the second, it is measured by probabilities based on a statistical model. Small as the difference may seem, it has far reaching consequences for both the

[5]For a more detailed description of the project, see Doherty *et al.* (2000) or `http://www.ida.liu.se/ext/witas/`

computational properties of the resulting predictive mechanisms and for the problem of model acquisition.

1.4 Overview

The survey and analysis of tasks involving prediction and the model designs they use are the topic of chapters 2 and 3, respectively.

Chapter 4 introduces the problem and context of our case study, including an overview of the sources of knowledge we can expect to be available when constructing a model instance. In chapters 5 and 6, we present and evaluate the two model designs. Chapter 7 describes an experimental environment, and experiences in constructing and using models of both designs in it, and chapter 8 concludes.

Part I

Chapter 2

Applications of Prediction

The first part of this thesis is a survey and analysis of predictive model designs, and applications in the context of which they appear. As we have noted, prediction is a ubiquitous activity, appearing in a wide range of reasoning tasks. The survey is, as it must be, selective, focusing on tasks that are most important to an artificial autonomous intelligent agent: perception, control, planning and diagnosis. Statistical forecasting, although more weakly related to those applications, is also included, because model designs and methods from this field have had a strong influence in many other areas.

2.1 State Estimation: Predictive Filtering

State estimation is the problem of constructing a world view, as correct as possible, from a sequence of partial and uncertain observations. Examples include positioning for a mobile robot, in which case the state is the robots position and observations may be sonar measurements of distances to surrounding objects, corrupted by noise, or tracking an aircraft, in which case the state is the position, velocity and acceleration of the aircraft and observations may be radar measurements of its position only, also disturbed by noise.

Assuming that disturbances are uncorrelated, the combined readings from several sensors yield a better estimate. Predicting the evolution of the observed system allows readings at different points in time to be combined, making more information available without the need for more sensors. This technique is known as *predictive filtering* or just filtering. Described in abstract, it works as follows[1]:

Let $f(\mathbf{x}(k) \,|\, \mathbf{y}(0) \ldots \mathbf{y}(k))$ be a probability density function, representing the state estimated from the k observations made so far. Upon receiving the $(k+1)st$ observation, the estimate is updated in two steps: First, a predicted state estimate is

[1]This formulation is taken from Pitt and Shephard (1999).

calculated as

$$f(\mathbf{x}(k+1)\,|\,\mathbf{y}(0)\ldots\mathbf{y}(k)) = \int f(\mathbf{x}(k+1)\,|\,\mathbf{x}(k))f(\mathbf{x}(k)\,|\,\mathbf{y}(0)\ldots\mathbf{y}(k)). \tag{2.1}$$

Second, the predicted estimate is corrected for the observation, according to Bayes rule,

$$f(\mathbf{x}(k+1)\,|\,\mathbf{y}(0)\ldots\mathbf{y}(k+1)) = \frac{f(\mathbf{y}(k+1)\,|\,\mathbf{x}(k+1))f(\mathbf{x}(k+1)\,|\,\mathbf{y}(0)\ldots\mathbf{y}(k))}{f(\mathbf{y}(k+1)\,|\,\mathbf{y}(0)\ldots\mathbf{y}(k))}. \tag{2.2}$$

The normalizing factor,

$$f(\mathbf{y}(k+1)\,|\,\mathbf{y}(0)\ldots\mathbf{y}(k)) = \int f(\mathbf{y}(k+1)\,|\,\mathbf{x}(k+1))f(\mathbf{x}(k+1)\,|\,\mathbf{y}(0)\ldots\mathbf{y}(k)), \tag{2.3}$$

is important since it measures how likely the sequence of observations as a whole is in the given model, and thus also provides a measure of how well the model fits reality.

The predictive model is simply an arbitrary probability density function

$$f(\mathbf{x}(k+1)\,|\,\mathbf{x}(k)), \tag{2.4}$$

as is the observation model[2]. All that is needed to turn this into a practically usable filtering procedure are finite representations of the probability densities involved, such that calculations can be carried out with reasonable efficiency.

2.1.1 The Kalman Filter

The Kalman filter (Kalman 1960) is a simple, mathematically elegant and very widely used filtering procedure. For a detailed treatment, see *e.g.* Maybeck (1979) or Grewal and Andrews (1993).

In a Kalman filter, the predictive model is a linear difference equation $\mathbf{x}(k+1) = A\mathbf{x}(k) + \mathbf{w}(k)$, where $\mathbf{w}(k)$ is a (vector valued) random variable representing noise, which is assumed to have Gaussian distibution with zero mean and a known covariance with equal power over all frequencies (the observation model is similarly

[2]Note that here the predictive model is memoryless, in the sense that the likelihood of a particular next state depends only on the present state. This property is known as the *Markov property*, and dynamical systems exhibiting it are called Markov processes. The abstract formulation of a predictive filter would of course admit a probability density conditional on several, or even all, past states, as well.

restricted). With these assumptions all involved densities are Gaussian, yielding finite representations as well as a simple and computationally efficient update. It can also be shown that under these assumptions, the estimate is optimal.

In practice, the strong assumptions of the Kalman filter are often not met. Most systems of interest are not linear, and even more rarely is noise perfectly white Gaussian. Techniques have been developed to cope with these problems, *e.g.* a shaping filter can be applied to make noise behave as white Gaussian, a non-linear model can be linearized around the current estimated mean or a bank of filters specialized for different regions of the state space can be used. What more, even in situations where assumptions fail to hold, the linear/Gaussian model is often a good enough approximation. Such models are also relatively easy to obtain through statistical estimation or a combination of physical modelling and estimation.

2.1.2 Particle Filters

Particle filters (Gordon, Salmond, & Smith 1993; Kitagawa 1996) represent an approximation of the estimate density by a finite sample, *i.e.* a set of weighted points ("particles"), making it possible to use any density function. The problem, however, lies in selecting the points following an update.

The relatively simple sampling/importance resampling (SIR) strategy requires only that the predictive model density (2.4) can be sampled for given values of the independent variable, and that the sensor model density can be evaluated for given values of both variables. Because the number of particles needed to approximate the estimate density well enough may be large, the method can be computationally expensive. Nevertheless, particle filter methods have been very successfully applied, *e.g.* for mobile robot localisation (see Thrun (2000) for a survey).

2.1.3 Discrete and Hybrid State Identification

State estimation problems arise also in discrete state spaces, in particular in the context of diagnosis[3]. In principle, this causes no problem since any probability density over a finite discrete space can be represented by enumeration. In practice, an enumerative representation is often computationally too expensive.

Thiebaux and Lamb (2000) describe a filtering method for hybrid systems, comprised of both continuous and discrete state components, and apply it to the localisation problem for a car travelling in a road network. The discrete part of the state is the sequence of straight road segments traversed by the car (the "segment history") and the continuous part its position along the last segment. For each discrete state with non-zero probability, a Kalman filter estimates the continuous state assuming the corresponding segment history. The probability of a change of discrete state is derived from the filter fit, as given by equation (2.3).

[3]Discrete state identification also appears in AI qualitative process theory, where it is called *measurement interpretation* (Forbus 1986; DeCoste 1990).

2.2 Reidentification

A problem closely related to state estimation is found in the process of anchoring symbolic referents to sensory perceptions (objects): Reidentification of a previously observed object.

The problem is particularly evident in visual tracking, where *e.g.* changing perspective or light conditions can cause the appearance of an object to change, when the object is temporarily lost from sight (or even between two consecutive frames, if the frame processing rate is too low), so much that comparing visual characteristics like shape and color is not sufficient to recognize it with any useful degree of certainty. Again, prediction can introduce a dependency between the past and present observations of the object, increasing certainty in the decision of whether to match the two.

Coradeschi and Saffiotti (2000), consider the problem as finding the best match, if any, for a previously anchored referent among a set of perceived objects, and call it reacquisition. The key point is that the observable signature of perceived objects should not be compared to the referents last observed signature, but to a prediction of what its signature should be in the present. Since Coradeschi and Saffiotti are defining an abstract framework for anchoring, the concrete form of the predictive model and the criteria for signature matching are left open.

In the work of Huang *et al* (1997; 1999) on tracking cars by means of cameras stationed along a highway, the predictive model takes the form of appearance probabilities that "define how an object observed at some point in the past can be expected to appear at some point in the future" (Huang & Russell 1997). They depend on both the object and the camera positions (for example, the objects estimated velocity and the distance to the next observation point determine the expected time of reappearance). The problem of tracking a vehicle by vision arises also in the WITAS UAV application. A prototype solution based on the Coradeschi-Saffiotti framework, incorporating only simple linear position prediction, has been experimented with (Coradeschi, Karlsson, & Nordberg 1999).

2.3 Control: Reaction and Anticipation

Although most feedback controllers, being simple and highly reactive devices, do not contain any form of model, modern controller design relies on models of the controlled system to select the most effective control law.

The traditional system model in control theory is a system of differential equations, which relate the rate of change of system state to the current state. Because the model is always a simplification of reality, the predicted and actual evolution from a starting state tend to diverge with time, but this uncertainty is not incorporated into the model. Instead, frequent corrective feedback to prevents the divergence from growing too large. The theory of controller design is most developed for linear system models, and control of non-linear systems is considered hard.

2.3.1 Model Predictive Control

Model predictive control (MPC) is a control paradigm that incorporates the system model, typically described by difference equations, in the controller. MPC theory and application are surveyed by Garcia *et al.* (1989).

In every iteration, the MPC algorithm computes, using the system model and measurements of the current system state, a control policy, trying to optimize the control objective, for a finite time ahead. This policy is executed, but only for a time much shorter than the prediction horizon. New measurements are then taken, and the process repeats.

There are a number of advantages to this method: non-linear system models cause less difficulty than in traditional control, the control objective can be more complicated, and so called hard constraints, *i.e.* constraints on the system state and control input that have to be obeyed, can be handled. Although the predictive model in MPC as in traditional control does not contain any explicit measure of its own inaccuracy, in MPC the discrepancy between predicted and measured system state model can be measured, and the model iteratively corrected on-line. There are also a number of drawbacks: the MPC algorithm requires more on-line computation than a static feedback control law, and therefore has seen most use in "low-speed" applications, *e.g.* refining or chemical process industry, where the control update cycle needs to be executed maybe once every minute, rather than once every microsecond (Qin & Badgwell 1997). Also, until recently, the theory of MPC, *e.g.* regarding stability, has been less well developed than for traditional control.

2.3.2 Anticipation in Tracking Control

Although tracking control, *i.e.* control problems in which the reference point is constantly changing, is inherently a reactive activity, predicting near future reference values can improve the precision and economy of the controller.

Taking a problem from the WITAS project, consider the UAV tracking a car traveling on a road: it is desirable to keep the UAV at some angle, say $25°$ - $45°$, behind the tracked car *w.r.t.* the car's direction of movement. Matching the expected future velocity of the car instead of that currently measured reduces the need for abrupt course corrections (reducing the risk of losing track of the car due to sudden camera movements) and the risk of overshoot. Even a very simple predictive model (essentially assuming that the car maintains its current x, y velocity) allows the UAV to approach the desired position and velocity much more smoothly. A more elaborate model could take into account the shape of the road and behaviour such as slowing down when approaching an intersection.

The principle applies on a larger time scale as well: if events in the near future can be predicted, more time is available to prepare a reaction and the actions taken can be better tailored to the developing situation. The Propice-Plan (Despouys & Ingrand 1999; Despouys 2000) task execution and monitoring system applies short-term prediction to make a more informed choice of which of several alternative procedures to apply to achieve a given goal, as well as to anticipate, and if possible avoid, potential failures before they become a fact.

2.4 Planning: Verification and Guidance

Planning, of all kinds, involves reasoning about the consequences of hypothetical action, *i.e.* prediction. In *e.g.* traffic planning, the hypothetical "action" considered may be the addition of a new link to the road network. Assessing the value of this action with respect to the planner's goal, be it to meet increasing travel demand or to move traffic away from certain areas for environmental reasons, requires predicting the situation that would result if the action was realised (the models used in traffic planning are examined in more detail in chapter 4).

2.4.1 Truth Criteria in AI Planning

Verifying a plan amounts to deciding if its execution, in a given initial situation, would indeed lead to the achievement of the intended goals. In many AI planning algorithms, *e.g.* regression or partial-order planning algorithms, this needs be done also for subgoals introduced in the planning process.

The "classical" AI planning problem assumes a finite, discrete set of states, in which actions cause deterministic transitions. The action and initial situation descriptions available to the planner are also assumed to be complete and accurate. Since there is no uncertainty, the prediction problem is in this setting extremely simple[4].

The methods of classical planning have been extended to problems with incomplete information and actions with random or nondeterministic results. Naturally, this makes the verification problem harder, as can be seen in that it contributes a significant part to the complexity of planning algorithms for such problems. For example, in CGP (Smith & Weld 1998), an extension of GRAPHPLAN (Blum & Furst 1997) to conformat planning[5], the solution extraction procedure is far more complicated than that of the classical GRAPHPLAN.

[4]Though note that if the steps of the plan are only partially ordered, determining what holds and what does not at a particular point in the execution of the plan is not as easy. This is the infamous Modal Truth Criterion of Chapman (1987), and is in general NP-hard to decide.

[5]In planning with incomplete information, a planner that builds unconditional plans, *i.e.* plans without sensing actions, is called conformant, while a planner that builds conditional plans, involving sensing actions or assuming complete observability, is called contingent.

2.4.2 Flaw Detection and Plan Repair

The iterative repair approach to planning consists in repeatedly detecting "flaws" in a current candidate plan and applying "repairs" to improve it. XFRM, CIRCA and WEAVER are particularly interesting examples of iterative repair planners, because they all make explicit use of prediction to detect flaws in candidate plans or to guide the selection of repairs.

The predictive model used by the XFRM planner (Beetz & McDermott 1992; McDermott 1994) consists of a discrete state space, defined by the set of truth assignments to a set of predicate instances, and two kinds of rules: event rules take the form

`while` $<$*cond*$>$ `with an average spacing of` τ `occurs` $<$*event*$>$

and specify that while in a state satisfying $<$*cond*$>$, $<$*event*$>$ appears at random according to a Poisson distribution with frequency $\frac{1}{\tau}$, while effect rules take the form

`if` $<$*cond*$>$ `then with probability` p $<$*event*$>$ `causes` $<$*effect*$>$`*`

with the obvious interpretation. The XFRM prediction model appears to be equivalent to a Markov jump process (see chapter 6), although this remains to prove formally.

Further work by Beetz *et al.* (Beetz & Grosskreutz 2000) extends the representation to models with non-discrete state, using hybrid automata (Alur, Henzinger, & Ho 2000). The state of a hybrid automaton contains both discrete and continuous components, and the values of the continuous components evolve over time by a function determined by the discrete state. Changes in discrete state are triggered by conditions on the values of the continuous state components. In (Beetz & Grosskreutz 2000), the evolution of the continuous state components is simulated by small, fixed-length time steps, and it's shown that by controlling the step length and the event frequencies arbitrary precision can be achieved.

In the CIRCA planning system (Musliner, Durfee, & Shin 1995; Goldman *et al.* 1997), the environment model is a timed transition system, containing both controllable and uncontrollable transitions. The planner constructs cyclic schedules for real-time execution, aiming to maintain safety by preventing the system from reaching failure states. For each candidate plan, possible execution paths are explored in a predicitve model that combines the environment model, a model of the execution system and the plan[6], and paths to failure states cause the plan to be revised.

The planning domain used by the WEAVER planner (Blythe 1998) combines classical, *i.e.* deterministic, actions with events that occur at random with specified probability. WEAVER interleaves contingent planning in a simplified domain, in which most of the possible events are ignored, and predicting likely failures in the resulting plan. Plan repair consists in partially "de-simplifying" the domain, by considering again the events that are most likely to cause the plan to fail, and replanning.

[6]The exhaustive exploration of the execution paths of a system is essentially *model checking*, a method developed in formal verification (Clarke, Grumberg, & Peled 1999). This, and timed transition system models will be examined more closely in chapter 5.

2.5 Probabilistic Reasoning and Decision Theory

The Markov decision process (MDP) is frequently used to model planning, or more generally, decision making, under uncertainty, *e.g.* (Kaelbling, Littman, & Cassandra 1998), (Blythe 1999). A detailed treatment of MDP theory, and many examples of applications, can be found in the book by Puterman (1994). Markov processes will also be examined in more detail in chapter 6.

There are four components to an MDP: a finite or countably infinite state space $\mathcal{X}$, a set of possible actions $\mathcal{A}$, a probability distribution $\mathbf{P}(X_{k+1} = x' \mid X_k = x, A_k = a)$ over successor states, conditional on the current state and action taken, and a reward function $R(x, a)$ of state and action. In every step of the process, the decision making agent selects an action and receives a reward depending on the state and the chosen action. The next state is then drawn randomly, conditioned on the current state and chosen action. This little game is repeated, either for a fixed number of steps or indefinitely. The fact that the successor state probability is conditioned only on the current state and action, not on the history of states and actions leading up to the current state, is known as the Markov property, giving the MDP its name.

A solution to an MDP is a mapping from state to action, $\pi(x)$, called a policy. For a given policy and starting state, the expected (in the statistical sense) reward can be calculated, *e.g.* as

$$\mathbf{E}(r^N_{\pi,x}) = R(x, \pi(x)) + \sum_{x' \in \mathcal{X}} \mathbf{P}(x' \mid x, \pi(x)) \mathbf{E}(r^{N-1}_{\pi,x}). \tag{2.5}$$

Equation 2.5 gives the expected total reward over the next N actions, but many other measures of expected reward can be defined, *e.g.* the discounted total or average reward per action, for a finite or infinite horizon, *etc.* The objective is to find a policy that maximizes the expected reward.

A Partially Observable MDP (POMDP) is an MDP with two additional elements: a set $\mathcal{O}$ of observations, and a probability distribution $\mathbf{P}(O_{k+1} = o \mid X_k = x, A_k = a)$ over observations, conditional on the current state and action taken. In a POMDP, the decision making agent does not know the current state of the process, only a probability distribution $\mathbf{P}(X_k = x)$ over the state space (this is often called a *belief state*). After the next state has been determined, an observation is drawn randomly, conditioned on the new state and the chosen action, and the observation is made known to the decision making agent, which may update its belief state according to equations (2.1) – (2.2), in the same way as a predictive filter. Because states are inaccessible, a policy for a POMDP must map belief states to actions, but besides this complication calculation of expected reward is the same as for an MDP.

As in the abstract predictive filter, the core of the predictive model in an MDP or POMDP is a conditional probability distribution. For small state spaces, the model (and the policy) can be described by enumeration, but to deal efficiently with large state spaces, representations or approximations that exploit structure seem to be necessary (Boutilier, Dean, & Hanks 1999).

2.5.1 Probabilistic Networks

Large state spaces typically arise when the state is comprised of several variables, each with its own value domain. The set of all possible states is then the cross product of all value domains, and thus grows exponentially. Probabilistic networks (Pearl 1988; Cowel *et al.* 1999) exploit this to provide a more compact representation of conditional probability distributions when many variables are independent, or only indirectly dependent. A probabilistic network consists of a graph whose nodes are the variables, $V_1, \ldots, V_n$, with arcs representing "influence" from one variable to another, and a conditional probability distribution over the values of each variable, $\mathbf{P}(V_i = v \mid \ldots)$. The advantage is that the distribution for a variable V_i needs to be conditioned only on the values of the subset of variables that directly influence V_i, *i.e.* the set of variables from which there are arcs to V_i.

Probabilistic networks appear to have been used mainly in expert systems and diagnosis (Cowel *et al.* 1999). Forbes *et al.* (Forbes *et al.* 1995) apply them to tracking and control problems in an autonomous vehicle.

2.6 Execution Monitoring and Diagnosis

Execution monitoring and diagnosis is often cast as the problem of detecting faulty behaviour in a system and isolating the malfunctioning system component responsible for it. The commonly adopted definition (*e.g.* Dean and Wellman (1991)) of "faulty behaviour" is behaviour that deviates from the expected behaviour[7]. Variations on the diagnosis problem can be distinguished *e.g.* depending on if observations are taken at one instant or over time, or what consitutes a diagnosis: a fault mode, a sequence of disturbances, a set of faulty components, *etc.* (de Kleer & Williams 1987).

Treated as a problem of detecting and identifying a fault mode, diagnosis is virtually indentical to state identification, when the (fault or non-faulty) mode of the system is viewed as part of the system state, *e.g.* Washington (2000) applies a model combining Kalman filters and Markov chains, similar to that of Thiebaux and Lamb (2000). It is in this area, in particular in the context of temporal diagnosis, that perhaps the widest range of different kinds of predictive models have been considered. Examples include discrete, probabilistic and hybrid transition systems (Williams & Nayak 1996; McIlraith *et al.* 2000), logical action theories (McIlraith 1998) and specifications in fuzzy linear temporal logic (Ben Lamine & Kabanza 2000)[8]. Statistical models, *e.g.* probabilistic networks, have also seen extensive use in diagnosis (Cowel *et al.* 1999).

[7]Saffiotti (1998) takes the, in a sense more general, view that monitoring consists not in deciding if the observed situation is expected, but whether the executing plan or program remains relevant in this situation.

[8]The logic (LTL) and progression algorithm used by Ben Lamine and Kabanza (2000) is similar to the logic MITL used in our expectation/normality model design (see chapter 5).

2.7 Statistical Forecasting

Applications of statistical prediction in *e.g.* business, engineering and medicine are too numerous to mention, and we have already encountered a number of examples. They all, however, have certain things in common: First, a statistical model takes the form of a probability distribution for the random variable to be predicted, also called the dependent variable, conditional on one or more indepedendent variables, or predictors. In most applications of statistical prediction, the distribution is assumed to belong to a particular family, *e.g.* linear functions with additive, normally distributed disturbance. Second, parameters of the distribution are estimated from data, in a statistically sound way. Characteristic of statistical prediction is that most computation is involved in the estimation of model parameters rather than in computing predictions from the finished model.

An important point to note is that statistical models are based on *correlation* between predictor variables and the dependent variable, and neither assume nor imply any causal relationship from predictors to the dependent variable. It may in fact be that predictor variables are caused by the predicted variable, as is the case in state estimation, or that variables of both kinds are caused by factors not present in the model at all.

2.7.1 Linear Regression

The linear regression model is perhaps the most common statistical prediction model, and should be found in any statistics textbook, *e.g.* Hoel (1984). Important reasons for the popularity of this model may be its relative simplicity, theoretically and computationally, and that it serves as a good approximation even in many cases where assumptions of the model do not hold.

Let Y be the dependent variable, and X the predictor (the case of multiple predictor variables is similar). The assumption is that X and Y have a joint normal distribution, which implies that the distribution of Y conditional on $X = x$ is normal with a mean that is a linear function of x, *i.e.* $\mathbf{E}(Y \mid X = x) = \alpha + \beta x$, and a variance, σ^2, that is independent of x. The relationship is often written in a slightly different form:

$$\mathbf{E}(Y \mid X = x) = \alpha + \beta(x - \mathbf{E}(X)). \tag{2.6}$$

From a pairwise sample of X and Y, $(x_1, y_1), \ldots, (x_n, y_n)$, least squared error estimates of $\hat{\alpha}$, $\hat{\beta}$ can be calculated. The estimates are maximum-likelihood, and since they are linear expressions in the sample of Y, they are normally distributed with variances determined by σ^2. For an unobserved value, x', of X, the predicted value of the Y is the expected value $\mathbf{E}(Y \mid X = x')$, calculated according to equation (2.6) using the estimated parameters $\hat{\alpha}$ and $\hat{\beta}$. The variance of the prediction is given by

$$\mathbf{V}(Y \mid X = x') = \sigma^2 \left(\frac{1}{n} + \frac{(x' - \overline{x})^2}{\sum_{k=1}^{n} (x_k - \overline{x})^2} + 1 \right), \tag{2.7}$$

and confidence intervals can be constructed by standard methods.

The strength of the correlation between X and Y is in a sense measured by the parameter β: if β is not statistically different from zero, X and Y are most likely not linearly correlated. In this case, the two variables may be uncorrelated, or their relationship may be one that can not be linearly expressed. Variance in the predicted value of Y is the sum of two sources of uncertainty: the first two terms in (2.7) are the variances of the estimates of the parameters α and β, and decrease with increasing size and span of the sample on which these are based, while the last term is the variance inherent in the distribution of Y. Note also that the second term in (2.7), which is the variance in the estimate of β, increases the further x', the unobserved value of the predictor, is from the mean of the sample.

2.7.2 Time Series

A form of statistical modelling and prediction of particular importance is *time series*, where data consists of observations of the same random variable over time: $X_{t_1}, X_{t_2}, \ldots$. As in all statistical modelling, the variable of interest is assumed to have a certain distribution and estimation is limited to the parameters of that distribution. Chatfield (1996) gives a good introduction to theory and practice.

Two kinds of models may be distinguished, depending on if the predictor for the dependent variable at a particular time t is the time t itself, or if it is the observed value of X at one or more previous time points. Models of the latter kind are often called *autoregressive*, and we have encountered several examples already, *e.g.* in predictive filtering (section 2.1) and Markov decision processes (section 2.5). In models with time as the predictor, X_t is often assumed to consist of three components: a trend, increasing or decreasing linearly with t, a seasonal variation, increasing and decreasing cyclicaly with t, and a random noise component, independent over time.

These models are of course highly simplified descriptions of the complex real world that generates time series data, and statistical modelling is typically "local", in the sense that uncertainty increases the further one extrapolates away from observed data. Chatfield remarks that "Prediction intervals, calculated on the assumption that the model fitted to past data will also be true in the future, are generally too narrow" (1996, p. 85). In practice, prediction from time series data often relies as much on the knowledge and intuition of the analyst as on the data.

Chapter 3

The Predictive Model

Without a model there can be no prediction. The knowledge, or assumptions, concerning the predicted system expressed in the model is essential for the prediction mechanism to be able to conclude something more precise about the future than that "anything can happen". Additionally, most model designs incorporate some measure of relative likelihood, so that of the future developments that are possible, some can be deemed more likely.

The previous chapter briefly described several different predictive model designs. In this chapter, we try to highlight some design choices, and issues raised by the various alternatives found. The chapter concludes with a discussion of how to evaluate model designs separately from their instantiation in concrete models.

3.1 Model Representation

The representation of the predictive model constrains the entire prediction mechanism, *e.g.* how a concrete model must be acquired, what results can be obtained from it and how those results are computed. Conversely, the choice of model representation is constrained by the results desired, the acquisition methods available, *etc.*

The discussion will be limited to model representations incorporating the notion of *state*, and, since we are interested in prediction, a state that evolves with time. Although any system can be viewed in this way[1], this is not reflected in all predictive model designs. For example, some models for state estimation and time series analysis omit the state and relate time directly to observable consequences of the state.

[1]In systems theory, it can be shown that for any general system a representation by state object and response function can be found (Mesarovic & Takahara 1975). Rosen (1991) argues that certain systems, in particular living beings, can not be understood in such a "mechanical" way.

3.1.1 Representation of State

What aspects of the state of the predicted system should be included in the model state and how they should be represented is largely determined by the constraints and needs of the application. An issue, however, that arises in several applications is the representation of infinite objects, *e.g.* a probability distribution over an infinite space. There seem to be at least three approaches: (*1*) representation by parameterized functions, (*2*) by constraints and (*3*) by a finite sample.

A parameterized function representation for probability distributions is used in *e.g.* the Kalman filter, where all distributions are Gaussian and thus describable by their first two moments, in statistical regression models, and to represent policies in some approximate techniques for solving POMDPs (Boutilier, Dean, & Hanks 1999).

Representation by constraints is not suited to probability distributions, but typically used in applications were all possibilities must be considered equally, *e.g.* to describe regions of continuous space in formal verification of timed and hybrid systems (Alur *et al.* 2000). Representation by a finite sample, on the other hand, is most useful for probability distributions, where frequency in the sample can be directly related to probability. An example seen in the previous chapter is the particle filter.

3.1.2 Representation of Time

Time is central in most kinds of predictive models. There seem, however, not to be so many different views to take regarding time:

(*1*) Time may be viewed as a continuum, in which the system state evolves continuously. The canonical example is a system of differential equations. Note, however, that simulation of such a system is in most cases based on approximating the continuous flow of time by small, discrete steps.

(*2*) Time may be viewed as a continuum, in which the state is for the most part steady and changes, instantaneously, only as a result of *events*, appearing at unequally spaced points. Examples of such models include timed and hybrid transition systems (Alur & Dill 1994; Alur, Henzinger, & Ho 2000), partially ordered plans and formalisms like the event calculus (Kowalski & Sergot 1986). The most common method of representing and reasoning with time in such models appears to be by some form of constraints.

(*3*) Time may be viewed as a sequence of discrete *points*, with a state at each point and state changes between points. This appears to be the most common view, examples including classical planning, difference equations used in state estimation and control, many diagnosis models, and more. This discrete time is often viewed as an abstraction of continuous change, and an implicit assumption is that the discretization is fine enough, *i.e.* that the changes between one point and the next amount to no more than the change from the state at the first point to the state at the next.

3.1.3 Representation of Uncertainty

In all but the most trivial cases, predicting the future is an uncertain business. At least three approaches to dealing with this uncertainty can be distinguished.

(*1*) Ignore uncertainty. In, for example, planning it is of course the case that any part of a plan always may fail (and fail in a number of different ways), but if failures are very rare, it may not be worthwhile to plan for the contingencies that failures cause and hence to view the expected outcome of every planned action as certain. In much the same way, although the models used in controller design will always deviate from the actual system behaviour, no estimate of this deviation is made within the model: instead, the prediction horizon is kept short by frequent corrective updates.

(*2*) Treat uncertainty as non-determinism, that is, to consider *all* possible outcomes regardless of likelihood. This is necessary when absolute certainty is required, *e.g.* in formal verification (Clarke, Grumberg, & Peled 1999), but is also adopted in contingent planning (Peot & Smith 1992; Collins & Pryor 1995; Weld, Anderson, & Smith 1998).

(*3*) Quantify the uncertainty. The most common way to quantify uncertainty is through probabilities, though alternatives exist, *e.g.* fuzzy logic. This approach is at its best when reliable measures or estimates, such as statistics, of any uncertainties are available, and when "sufficient" or "best possible", rather than absolute, certainty is required of the prediction. Examples are found in all areas surveyed in the previous chapter.

From an algorithmic point of view, the difference between non-deterministic and probabilistic representations seems to be small. For example, dynamic programming methods for solving MDPs translate into algorithms for non-deterministic planning by considering at each step the worst outcome rather than each outcome weighted according to probability (Koenig & Simmons 1995; Bonet & Geffner 2000). There is, however, a difference in the interpretation of the result. For example, a *strong cyclic plan*, as defined by Cimatti *et al.* (1998), is guaranteed to reach its goal eventually, assuming no possible execution path of the plan is ignored forever. In other words, the plan works if any uncertain action outcomes are selected at random (with non-zero probability for each outcome) but may loop forever if action results are non-deterministic.

3.2 Model Acquisition

For a given system and a chosen model design, we term the process of constructing a concrete model instance reflecting the system *model acquisition*. We can distinguish at least three approaches to model acquisition: (*1*) modelling from first (or physical) principles, (*2*) estimation (or learning) from data, and (*3*) encoding of "expert knowledge". Two or more approaches are often combined, *e.g.* in modelling dynamical systems by differential or difference equations, *e.g.* for control design, the form of the relationship can often be derived from physical laws, while actual numerical coefficients are best estimated from data (Ljung 1987).

Representations and modes of acquisition are also matched to varying degree. For example, many learning methods require a representation to which small, incremental changes can be made. There are well-known and theoretically well-founded statistical methods for estimating probabilities from data, while people, experts or not, are notoriously poor at estimating probabilities (Tversky & Kahneman 1974)[2]. Perhaps most important in the choice of model acquisition method, and therefore indirectly in the choice of model representation, is to match the knowledge required by each mode of acquisition with the knowledge sources available.

3.3 Computation

The purpose of the predictive model is to *compute* predictions. However, the importance of computational properties such as scaling, numerical stability or even decidability depends on the application, specifically on how predictions are to be used.

For example, contingent planning is in most cases restricted to relatively simple forms of plans, typically trees of finite depth, because every additional complication in a plan makes the problem of verifying it, for every possible outcome, immediately much harder. The XFRM planner, on the other hand, never considers every outcome but only a random sample, and therefore can allow much more complex plan forms. Another example is the model representation developed in chapter 5. This representation has its roots in formal verification, but relaxes several restrictions normally made. The reason is that those restrictions are necessary to make verification problems, which concern the infinite horizon behaviour of the modeled system, decidable, while the application we are interested in is concerned only with finite predictions.

More generally, we can from the computational point of view distinguish predictions of at least three different kinds: (*1*) A more or less explicit enumeration of all possible developments up to the prediction horizon, or states at the time of interest (this, of course, includes probability distributions over states). (*2*) An, in some sense, representative sample of developments or future states, picked at random. The selection may be according to probability, but need not be: in some applications, *e.g.* formal verification, it is interesting to consider the states that are extreme *w.r.t.* some condition. (*3*) Answers to questions about the development or future state of the predicted system, *e.g.* if a state satisfying some condition can possibly occur, will necessarily occur, or what the probability of such a state occuring is. Most such questions, however, would seem to require either a more or less complete enumeration of possible developments or representative sampling.

[2]In expert system design, it has been argued that probability estimates provided by experts should not be interpreted as probabilities in the strict, statistical sense, but rather as some "measure of confidence", which happens to behave exactly according to the laws of probability (Cowel *et al.* 1999).

3.4 Evaluating Model Designs

Searching for criteria to evaluate or compare different predictive model designs, it is tempting to suggest "accuracy". This, however, would be a mistake since the accuracy of predictions is a product of the concrete model instance, and only indirectly influenced by the model design: although a good model design may permit the construction of more accurate model instances, it is certainly possible to construct a bad model using even the best design.

Davis *et al.* (1993), state that a knowledge representation fulfils several different roles: it embodies a set of ontological commitments, and a notion of what constitutes "correct inference", and is at the same time both a tool for computation and a medium of human expression. Thus, comparison of knowledge representations should be made from several points of view. Likewise, in comparing model designs, we suggest looking at three points:

(*1*) The adequacy and efficacy of the model representation: *i.e.* is the representation rich enough to express all the knowledge that is available? (or, in other words, can the best possible model be framed in this representation?) The dual question, is the available knowledge sufficient to construct an instance of the model design? is equally important.

Related questions are the difficulty, and cost, of constructing, and maintaining, the model. Representational properties such as modularity and transparency may be important if the model is to be maintained over time.

(*2*) Computational efficiency: *i.e.* can a prediction algorithm based on the model design be made efficient enough for the needs of the application?

(*3*) Integration with the application: does the model design provide predictions of the kind and quality that the application needs? This is, arguably, the most important point, since the value of prediction lies in its use in a larger context.

None of the questions raised above are simple enough to have a single, generally valid answer. The relative importance of each question depends, naturally, on the application.

3.4.1 Empirical Evaluation

In the end, the merits of a predictive mechanism in the context of a particular application comes down to how much it contributes to the performance of the system that it is part of, *i.e.* the "value added". In most cases this can only be determined empirically, by constructing the best model that available knowledge, time and resources allows and testing its impact on the system. While this may seem like an overwhelming amount of work only to assess the model design, it may, from the application point of view, be worthwhile if the predictive mechanism does improve on performance.

However, to compare different model designs empirically requires the completely unbiased construction of model instances of the designs under investigation, making it an even more labourious and difficult task, and perhaps less valuable from the application perspective. Nevertheless, it is something along these lines that we will do in the second part of this thesis.

Part II

Chapter 4

The Vehicle Movement Prediction Problem

The second part of this thesis is a case study, concerning a particular prediction problem. In this chapter, we describe the problem, the context in which it arises, and the knowledge sources we can expect to be available in constructing a predictive model, and sketch briefly some possible model designs. Chapters 5 and 6 develop two different model designs and associated algorithms in detail. The two designs are compared in chapter 7.

4.1 Problem Context: The WITAS UAV

The aim of the WITAS project is to develop technology for enhanced autonomy, applicable to a wide range of systems. The development of a UAV for traffic surveillance may be viewed as a "pilot study": it presents a challenge for state-of-the-art computer vision, control and reasoning techniques, but is still in the realm of the possible.

4.1.1 The UAV

The UAV is a miniature helicopter. Two different platforms are being used in the project: an APID Mk. III, manufactured by Scandicraft Systems AB[1], and a Yamaha RMAX, made by Yamaha Motor Company Ltd[2]. Figure 4.1 shows the RMAX with camera housing mounted underneath. Both helicopters measure around 3 meters in length and rotor diameter. The RMAX carries a payload of approximately 30 kilograms, the APID somewhat less.

[1]`http://www.scandicraft.se/`

[2]`http://www.yamaha-motor.co.jp/sky-e/rmax.html`

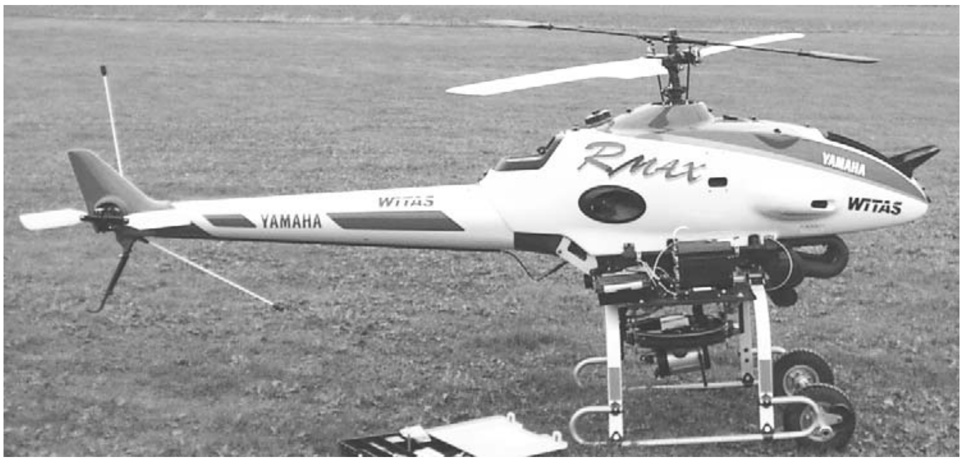


Figure 4.1: The Yamaha RMAX UAV.

The primary means of observing the ground is an active vision system, fed through a video camera mounted in a stabilized gimbal housing underneath the UAV. The vision system also has access to very accurate estimates of the position of the UAV and the orientation of the camera, as well as to a rich geographic data base. Nevertheless, due to video quality and the constraint of finding and tracking moving vehicles in real-time, not many identifying traits can be discerned: at best, we can hope to have good estimates of the position, velocity and size of the vehicle, and rough estimates of its shape and color.

In the traffic surveillance application, typical tasks for the UAV include locating and tracking a specific vehicle, or monitoring traffic in an area and looking out for particular situations or events. Other uses for the UAV are also investigated: one interesting application is photographing an area, *e.g.* for the purpose of mapping or constructing a detailed simulation environment, covering buildings and other significant structures from all points of view.

4.1.2 Prediction Problems in UAV Tasks

Several prediction problems arise in the control, sensory and reasoning tasks performed by the UAV, for example in reidentifying and tracking ground vehicles, and in on-board system monitoring and diagnosis. A few examples were described in chapter 2.

We focus on the problem of predicting vehicle movements on a relatively large time scale. The problem arises in the context of at least two different tasks, described below[3].

[3]In the scenario illustrations, we allow ourselves to become slightly science fictionish.

Reidentifying a Vehicle

Given the limitations of the vision system, and given the fact that most cars tend to look a lot alike, determining if a newly observed vehicle is in fact the same as one seen a while ago is a difficult problem (obviously, reading license plates is not an option). As described in section 2.2, to answer the question it is often helpful to determine the likelihood that the suspected same vehicle would be at this location at the present time, given the location and time it was last observed.

Scenario 1
The UAV is hovering over a city street, with a clear view for many blocks up and down the street, but a very limited view of the side streets. A black van appears making a right turn onto the street, which it follows for a block before turning right again. About two minutes later, a black van appears again, in the same place as the previous, making a right turn around the corner. Is it the same car?

What we would expect from prediction in this scenario is the answer that this is possible, if the time between its disappearance and reappearance is reasonable given the distances and density of traffic, but only if the driver of the van is circling, which is unusual. It could be considered normal if the driver is looking for a place to park, but this hypothesis can only be accepted if there are no free parking places in sight. The hypothesis also leads to the further prediction that the van currently in view, if it is the same car, will not turn right again at the same corner this time.

Planning a Search Strategy

A typical task for the UAV may be to find a specific vehicle, based on a description or previous observation: this requires planning a search strategy. Assuming that the vehicle is behaving independently of the UAV, *i.e.* that it is neither cooperative nor adversarial, the problem can be decomposed into two parts: (*1*) determine locations and times where the vehicle is most likely to be intercepted, and (*2*) plan a flight path to cover as many of those locations and times as possible[4].

Scenario 2
A short time after the second appearance of the black van, the ground operator notices it and instructs the UAV to track it. But the UAV, having had no special interest in it, no longer has the black van in sight.

Here, we expect prediction to indicate that the van should have followed the street it was on at least up to the next intersection, and there either continue or turn

[4]The problem appears similar to the "submarine hunt" problem studied in search theory (Morse 1978). The main difference is that while the submarine may move in any way it is capable of and geography allows, we assume that the vehicle conforms to "normal car behaviour". Although the submarine is "adversarial", in the sense that it is trying to evade its pursuer, this is usually disregarded, based on the assumption that the submarine has no way to observe the pursuers strategy, and therefore no way to adapt its behaviour in response.

left. Assuming the time since the last observation is short enough, that limits the possible current locations to two road sections, and the UAV can begin to plan how to reposition itself so as to have a view of both.

4.2 Vehicle Movement Prediction

The scenarios in the preceeding section illustrate a need to predict the location of a vehicle travelling in a road network. Two things conspire to make this problem very hard: predictions have to be made on a large time scale, on the order of several minutes (which in an urban road network allows for a lot of uncertainty), and quite scant information about the vehicle can be expected to be at hand (one or a few observations of previous location at most). For the problem not to be unsolvable, we have to settle for predictions that are not very exact, but fortunately, in both scenarios, knowing which road, or road section, the vehicle is likely to be in is enough.

We have chosen this particular problem because it is one of the most "knowledge intensive" prediction problems found in UAV tasks: due to the time scale, the behaviour and intentions of the driver, and the surrounding traffic situation, play a far greater role than vehicle dynamics in determining where the vehicle is going to be found. The strong influence of such unobservable factors means that a predictive model has to be built largely on knowledge about average cases and default assumptions, and therefore makes the knowledge representation aspect of the model design important.

4.2.1 Available Knowledge

We can reasonably assume that the UAV on-board geographic database contains detailed information about the road network: the physical network shape, legal speeds, one-way restrictions, traffic lights and other regulatory measures, *etc.*

While very little may be known about the particular vehicle whose movements are the subject of prediction, there is a large amount of knowledge about average or "normal" vehicle behaviour. A source of such knowledge is statistics, to the extent that it is available for the area of interest. Most interesting from our perspective are (*1*) the time it takes a vehicle, on average, to travel along a link in the network and (*2*) how the volume of traffic travelling from one node to another distributes over the available routes. We examine statistics and models commonly used in traffic planning in section 4.3.

An alternative, or complementary, approach is to make use of the substantial "common sense knowledge" possessed not only by experts but everyone who regularly drives a car. However, the reliability of such knowledge is uncertain, and it may also be difficult to elicit.

4.2.2 Related Work

Prediction and the representation of knowledge are common problems, and there is a wealth of work related to both. Concerning the particular problem of predicting vehicle movements, however, there is as far as we are aware no previous work.

Most closely related is probably the work of Forbes *et al.* (1995), on predicting short-term movements of a car (lane change, speed change, *etc.*) relative to other cars in the close vicinity. Their model is a probabilistic network: this is possible since, due to the short prediction time scale, they can take a step-based view of time.

4.3 Traffic Statistics and Planning Models

Traffic is a well studied subject, due to its importance for city and land development planning, and lies in the intersection of many scientific areas, *e.g.* economics, social and environmental studies, and operations research. Measurements of road use, average vehicle speed, *etc.*, provide a picture of current traffic patterns in a road network and may indicate where problems are. Traffic planning models are tools used to predict future travel demand and the impact of control measures, such as construction of new roads, imposing road tolls, *etc.*

4.3.1 Road Networks and Traffic Flows

In the study of traffic, the system consisting of vehicles and road network is typically simplified into a *network flow* representation: the road network is represented as a graph of *nodes* and directed *links*, and traffic as *flow* along links. Nodes of the network can represent intersections, where links meet, but also entry/exit points to the network, such as parking places, and may therefore have a net flow not equal to zero, to represent journeys starting and ending at the node.

At least four kinds of statistics are frequently used to describe the flow of traffic across the network:

Link flows

The *link flow* is the number of vehicles traversing a link per some unit of time. A common form is the "yearly day average", *i.e.* the number of vehicles per day, averaged over a year (Matstoms, Jönsson, & Carlsson 1996). Naturally, the flow varies during the day, and in many cases also during the year. The flow in number of vehicles per hour, for a given hour of the day and a given month of the year, is calculated by multiplying the yearly day average by a month/hour *index* for the link or link category (and dividing by 24). Figure 4.2 shows the month/hour index for an average Swedish countryside road.

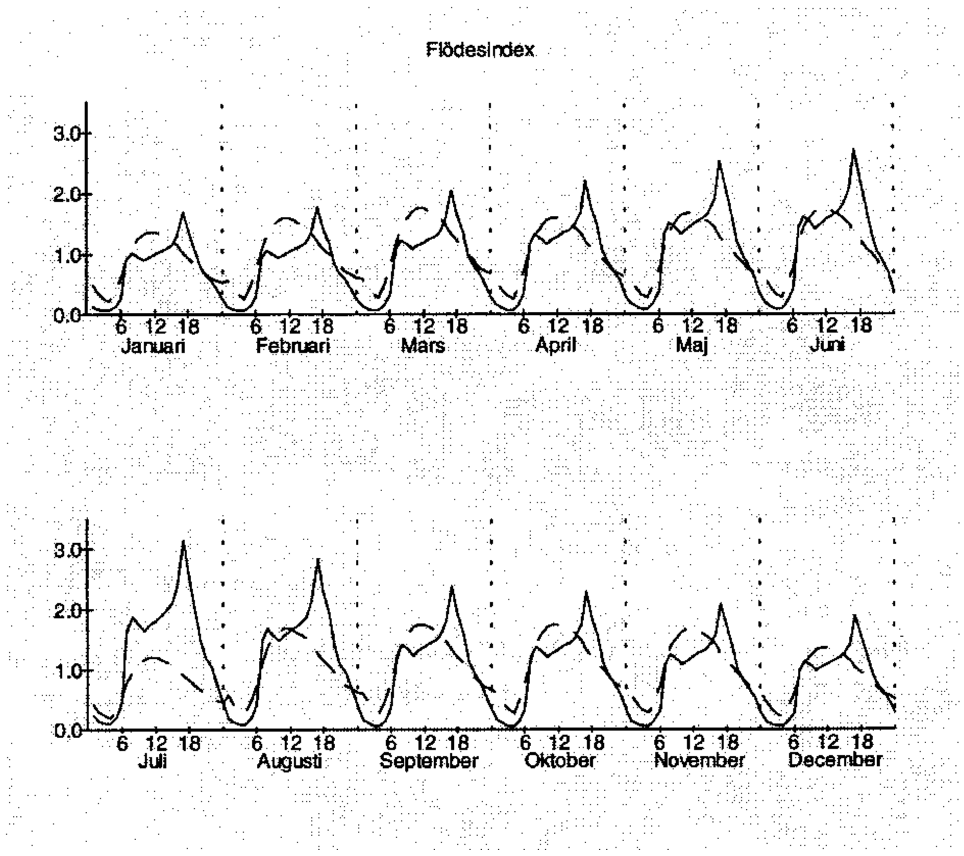


Figure 4.2: Month/hour flow index for Swedish countryside roads. The dashed line represents heavy traffic (trucks *etc.*). Reprinted from Matstoms, Jönsson & Carlsson (1996).

Link flows are often measured only for a selected set of links in a city or region, and it is in general not possible to calculate flows in remaining, unmeasured links. The network must satisfy the conservation of flow principle, *i.e.* for any node the sum of flows on incoming links, plus the volume of vehicles starting from the node, equals the sum of flows on outgoing links, plus the volume of vehicles ending their journey in the node:

$$\sum_{l:(n',n)} f_l + O_n = \sum_{l:(n,n')} f_l + D_n \qquad \forall n \in \mathcal{N} \tag{4.1}$$

Even if many nodes are assumed to be "non-origin/destination", *i.e.* such that $O_n = D_n = 0$, the network flow remains in most cases severely underconstrained.

Origin/Destination Counts

The *origin/destination* (O/D) *count* for a pair of nodes (m, n) is the number of vehicles, per some unit of time, travelling from m to n, and reflects the demand for travel (by car) between different places. Usually only a subset of network nodes are considered to be origin and/or destination nodes, while remaining nodes are assumed to have zero-sum flow.

Current O/D counts may be measured by surveying, but in long-term traffic planning, where one is interested in the future travel demand, they are typically predicted from social and economic factors, such as population, employment, car ownership, *etc.*

Route Flows

Between any pair of nodes (m, n), there are typically many possible routes through the network. The *route flow*, *i.e.* the distribution of the volume of vehicles travelling from m to n over the available routes, is the most fine-grained measure of traffic flow used.

Route flow distributions are, as far as we are aware, never measured, and to do so seems very complicated and expensive[5]. Instead, they are calculated from travel time functions and O/D counts using the traffic assignment model (this is described in section 4.3.2 below).

Link Travel Time Functions

Normally, the time it takes a vehicle to traverse a link is approximately the link length divided by the link speed limit, but as the volume of traffic increases, congestion

[5]This may actually be an application for the WITAS UAV, since a way to estimate route flows would be to track vehicles, chosen at random, from origin to destination. Doing this by UAV just might make it cheap enough to make statistically relevant sampling feasible.

effects cause the average speed to go down, and the travel time therefore to increase. Thus, the average link travel time (or equivalently, the average link speed) is a function of the link flow, often called a *volume/delay* (V/D) *function*. Many types of functions are used, but they generally have a common shape: up to a certain volume (called the *free flow capacity* of the link) travel time is constant, then increasing with increasing flow.

V/D functions are typically estimated from measurements, but only for a representative set of links. Matstoms *et al.* (1996) present V/D functions for 70 road types, covering most roads in Sweden.

4.3.2 The Traffic Assignment Problem

Calculating the distribution of traffic across available routes in a road network is done in one or more stages, depending on what measures are available as input: if O/D counts are not known, the demand for travel between different regions, and how this demand distributes over different means of travel (private car, public transport, *etc.*) must be estimated (or predicted). The final stage, known as the *traffic assignment problem*, is the most mathematically oriented, and also the most relevant to our problem. A good introduction to the subject is given by Patriksson (1994).

Input to the problem is the node/link representation of the road network, O/D counts for some subset of nodes, and travel time functions for each network link. Solutions to the traffic assignment problem are based on two fundamental assumptions: (*1*) that every driver minimizes his or her own travel time, and (*2*) that the situation remains stable long enough for the system to settle into an equilibrium state. At equilibrium flow, for any O/D pair, travel times along the routes actually used are all equal and less than any alternative route, and the average travel time is minimal[6].

Given a road network with associated travel time functions and O/D counts, the equilibrium flow can be obtained by solving a non-linear optimization problem:

$$\begin{array}{rcl} \min \sum_{l \in \mathcal{L}} \int_0^{f_l} T_l(v)\,dv & & \\ \sum_{r \in \mathcal{R}_{mn}} h_{mnr} & = & Q_{mn} \quad \forall (m,n) \in \mathcal{N}^2 \\ h_{mnr} & \geqslant & 0 \quad \forall (m,n) \in \mathcal{N}^2, r \in \mathcal{R}_{mn} \\ \sum_{(m,n) \in \mathcal{N}^2} \sum_{r \in \mathcal{R}_{mn}, l \in r} h_{mnr} & = & f_l \quad \forall l \in \mathcal{L} \end{array} \tag{4.2}$$

where $\mathcal{N}$ and $\mathcal{L}$ are the set of nodes and links, respectively, T_l is the travel time function associated with link $l \in \mathcal{L}$ and $\mathcal{R}_{mn}$ and Q_{mn} are the set of routes (we write $l \in r$ to indicate that link l is part of route r) and the O/D count, respectively, between nodes n and m. The variables f_l and h_{mnr} represent the total flow on

[6]This characterization is known as *Wardrop's conditions.*

link l and the flow from node m to n taking route r, respectively. Several other formulations, most of them more efficient, are possible, but (4.2) is the, perhaps, most intuitive.

4.3.3 Equilibrium Flow as a Model for Prediction

In traffic planning, the main use for equilibrium route flow distribution is to discover potential problems, such as frequent congestions, traffic loads causing environmental strain, and so on. For vehicle movement prediction, equilibrium flows may be used to assign likelihoods to different route choices, given destination, or to different destinations, given one or more observations of choices of route. Underlying this use is the correspondence between probability and frequency, in this case the fraction of the total volume of traffic taking a particular course and the probability of an individual vehicle taking that same course.

If the true route flow distribution is known, this is correct, and in fact the best that we can do. But the route flow distribution calculated as the solution to a traffic assignment problem rests not only on data but also on certain assumptions: that every driver minimizes travel time, and that every driver is well-informed enough to do so optimally. These assumptions are quite likely too strong in reality[7]: the problem from our point of view is what likelihood to assign the suboptimal route choices, which are assigned zero probability by the direct correspondence model.

Link flows combined with travel time functions yield average travel times for each link in the network, which is obviously useful for prediction. Again, however, estimates of the distribution of deviations from the average are missing.

4.4 Possible Model Designs

Due to the large time scale and the scarcity of knowledge about the vehicle of interest, any prediction made will be highly uncertain: there are simply too many possibilities, and most of them are, more or less, equally likely. The only tool at our disposal to reduce this uncertainty is knowledge about average or normal vehicle behaviour, and therefore the representation and use of that knowledge is an important point in our model design. As noted above, we have knowledge of two kinds, statistical and "common sense", and different representations are more suitable for each kind.

The sheer mass of possibilities also means that we can not examine them all, unless we adopt a high level of abstraction. Because a large part of the uncertainty concerns time, an abstract state representation is naturally combined with an event-based view of time. Alternatively, we may adopt a more detailed state representation

[7]They seem, however, to yield close enough approximations to be useful for planning purposes. Florian and Nguyen (1976) found very good agreement between predicted and measured link flows and travel times, but also that predictions were very sensitive to small adjustments to the travel time functions.

and a continuous or step-based view of time, and use only random sampling among possible developments.

We develop two model designs. Both have an abstract state representation in which the position of a vehicle is described at the network level, *i.e.* only by what link it is in. Because of the uncertainty and large differences in the time the vehicle stays in a link, both designs are event-based. The main point of difference is the representation of uncertainty and how the relative likelihood of different developements is determined: in the first, it is a "qualitative" measure based on a heuristic model of expectations, while in the other, it is measured by probabilities and based on the traffic models discussed above. Note, however, that we can take some advantage of available statistics also in the construction of the first model.

Chapter 5

Expectation/Normality Models

In this chapter, we develop the first of our two model designs, based on the timed transition system formalism: model states are discrete, representing a high level of abstraction, and the model dynamics are event driven.

The most important characteristic of this design, however, is that the relative likelihood of different developments is a qualitative, heuristic measure based on a hierarchy of *expectations*. Expectations are rules, or schemata, that hold "most of the time", *i.e.* they express the *normal* case. Consequently, a development is normal to the degree that it satisfies expectations in the hierarchy, and we interpret this normality as likelihood.

Likelihood in this sense, however, is not probability: first, normality is a much coarser measure, and second, and more importantly, the normality measure is not numerical, but only a scale. For instance, a development that satisfies twice as many expectations is not twice as likely. Likewise, if two developments satisfy the same expectations, they are not necessarily equally likely, in a probabilistic sense: rather they are incomparable, *i.e.* equal only in the sense that they can not be ordered.

Expectations are represented by formulas in MITL, a temporal logic. A formula in MITL expresses a constraint on a timed, infinite sequence of states, which is precisely the form that the possible developments, generated by the timed transition system part of the model, take. To determine the normality of a development thus becomes essentially a problem of model checking MITL formulas, although over finite sequence prefixes. For this, we adapt the progression algorithm of Bacchus and Kabanza (1996).

5.1 Technical Background

This section briefly introduces timed transition systems and Metric Interval Temporal Logic (MITL). The roots of both formalisms lie in research on formal specification and verification of reactive hardware/software systems. Detailed introductions can be found in *e.g.* Alur (1999) and Emerson (1990), respectively.

Timed transition systems (Alur & Dill 1994) are essentially finite state automata, augmented with two kinds of time constraints: (*1*) each transition has a time window in which it is possible to make the transition and (*2*) each state has a maximal time (which may be ∞) that the system may remain in the state before it has to exit by some transition.

Let $\mathbb{R}^+$ denote real numbers $\geqslant 0$, with a special symbol ∞ for infinity.

Definition 5.1 (Timed Transition System)
A *timed transition system*, $S = (Q, R, C, L)$, consists of a set of *states* Q, a *transition relation*

$$R \subseteq \Sigma \times Q \times Q \times \mathbb{R}^+ \times \mathbb{R}^+$$

where Σ is some set of *event* labels, a *state constraint* function $C : Q \longrightarrow \mathbb{R}^+$, and state labelling function $L : Q \longrightarrow 2^P$, where P is some set of propositional symbols (often the set of states is 2^P, *i.e.* a state is defined by its properties).

As usual, if $(a, q, q', t, t') \in R$, the system may transit from state q to q' in response to the event a, but only in the time interval $[t, t']$ relative to the time that the system entered q (representing time constraints of the first kind). The system may remain in state q for a time at most equal to $C(q)$ (representing time constraints of the second kind).

Like a finite automaton accepts a set of strings over its alphabet, a timed transition system "accepts" a set of state/event histories: A *development*, (d, T), consists of a sequence of states and events marking state transitions, $d = q_0, a_0, q_1, a_1, \ldots$, and a function $T : d \longrightarrow \mathbb{R}^+$ that tells the starting time of each state (without loss of generality, $T(q_0) = 0$ may be assumed). The time of an event is identical to the starting time of the state it initiates, *i.e.* $T(a_i) = T(q_{i+1})$. We denote the duration of a state q_i by $D(q_i) = T(q_{i+1}) - T(q_i)$.

A development is a trace of the timed transition system S iff it satisfies the following conditions:

(*i*) for $i \geqslant 0$, there exists $t, t' \in \mathbb{R}^+$ such that $R(a_i, q_i, q_{i+1}, t, t')$ and $T(q_i) + t \leqslant T(q_{i+1}) \leqslant T(q_i) + t'$, and

(*ii*) for $i \geqslant 0$, $T(q_{i+1}) \leqslant T(q_i) + C(q_i)$

Two additional properties are usually required of a timed transition system: *executability*, which is the requirement that any finite prefix satisfying conditions (*i*) and (*ii*) can be extended to an infinite development, and *non-zenoness*, which is the requirement that the system does not make an infinite number of transitions in finite time. Because we are only concerned with finite developments (in time and number of events), these properties are not important to us.

Definition 5.1 is slightly unorthodox: the standard way to define timed transition systems is by extending finite state automata with a set of real-valued *clocks* that increase uniformly with the passing of time, and associating clock constraints and reset actions with transitions and states. The reason we have chosen to define timed transition systems in a different way is modularity of representation, as will become clear in section 5.2.

5.1.1 Metric Interval Temporal Logic

The Metric Interval Temporal Logic (MITL) is a so called "tense modal logic", and was developed as a language for specifying properties of real-time, reactive systems (Alur, Feder, & Henzinger 1996).

Definition 5.2 (MITL Syntax)
The language of propositional MITL consists of a set of atoms P, representing properties of states, propositional connectives and four temporal operators: $\Box_{[t,t']}\varphi$ (*always* φ), $\Diamond_{[t,t']}\varphi$ (*eventually* φ), $\bigcirc_{[t,t']}\varphi$ (*next* φ) and $\varphi\mathcal{U}_{[t,t']}\,\psi$ (φ *until* ψ). The intervals adjoined to the operators express metric temporal restrictions, and take point values in $\mathbb{R}^+$.

In MITL as defined by Alur *et al.* (1996), there is no *next* operator, and singleton intervals, *i.e.* of the form $[t,t]$, are not allowed. The restrictions are necessary to make certain questions concerning infinite developments decidable, and again, because we are only concerned with finite developments, we ignore these restrictions.

Formulas in MITL are evaluated over an infinite timed development (d,T). Since MITL formulas only reference present and future states, any suffix of a development is, for the purpose of evaluating formulas, also a development. When we say a formula holds in a state q_i, we mean in fact that it holds in the development suffix beginning with q_i.

Definition 5.3 (MITL Semantics)
Let d^i denote the suffix of d starting with the ith state.

A formula φ not containing any temporal operator holds in d^i iff φ evaluates to T in the state q_i. The truth conditions for temporal formulas are

- $\Box_{[t,t']}\varphi$ holds in d^i iff φ holds in every d^k such that $T(q_i)+t \leqslant T(q_k) \leqslant T(q_i)+t'$ or such that $T(q_i)+t < T(q_{k+1}) \leqslant T(q_i)+t'$.
- $\Diamond_{[t,t']}\varphi$ holds in d^i iff there exists an q_k such that $T(q_i)+t \leqslant T(q_k) \leqslant T(q_i)+t'$ or $T(q_k) < t < T(q_{k+1})$, and such that φ holds in d^k.

- $\bigcirc_{[t,t']}\varphi$ holds in d^i if φ holds in q_{i+1} and $T(q_i) + t \leqslant T(q_{i+1}) \leqslant T(q_i) + t'$.
- $\varphi\,\mathcal{U}_{[t,t']}\,\psi$ holds in d^i iff there exists a q_k such that $T(q_i)+t \leqslant T(q_k) \leqslant T(q_i)+t'$ or $T(q_k) < t < T(q_{k+1})$, ψ holds in d^k and φ holds for all d^j with $i \leqslant j < k$. begins to hold.

Connectives are interpreted as in ordinary logic.

The extension to first order MITL is straightforward.

5.1.2 The MITL Progression Algorithm

Because we will deal only with finite development prefixes, we need a way to evaluate MITL formulas incrementally. The MITL progression algorithm, due to Bacchus and Kabanza (1996), provides precisely this. The algorithm takes as input a MITL formula φ and a state q_i with duration $D(q_i)$, and returns a formula φ' such that φ holds in q_i iff φ' holds in state q_{i+1}. The algorithm, however, makes no reference to state q_{i+1}, or any other future state, and can therefore be applied to a finite development prefix.

Algorithm 5.4 (MITL Progression)
Let φ and q be the input formula and state, respectively, and φ' the returned formula. The progresion algorithm works recursively, by cases depending on the form of the input formula:

(i) If φ contains no temporal operators (we call such a formula a *state formula*), $\varphi' = \text{TRUE}$ if φ is true in q and $\varphi' = \text{FALSE}$ if not (note that TRUE and FALSE are syntactic constants, not semantic values).

(ii) If φ combines one or more subformulas with a propositional connective, φ' is the result of likewise combining the result of progressing each subformula. For example, if $\varphi = \alpha \wedge \beta$ then $\varphi' = \alpha' \wedge \beta'$.

(iii) If $\varphi = \bigcirc_{[t,t']}\psi$, then

  (iii.a) if $D(q) < t$ or $t' < D(q)$, then $\varphi' = \text{FALSE}$, and

  (iii.b) if $t \leqslant D(q) \leqslant t'$, then $\varphi' = \psi$.

(iv) If $\varphi = \Box_{[t,t']}\psi$, then

  (iv.a) if $D(q) \leqslant t$, then $\varphi' = \Box_{[t-D(q),t'-D(q)]}\psi$,

  (iv.b) if $t < D(q) \leqslant t'$, then $\varphi' = \psi' \wedge \Box_{[0,t'-D(q)]}\psi$, and

  (iv.c) if $t' < D(q)$, then $\varphi' = \psi'$.

  where ψ' is the result of progressing ψ.

(v) If $\varphi = \Diamond_{[t,t']}\psi$, then

$(v.a)$ if $D(q) \leqslant t$, then $\varphi' = \Diamond_{[t-D(q),t'-D(q)]}\psi$,
$(v.b)$ if $t < D(q) \leqslant t'$, then $\varphi' = \psi' \vee \Diamond_{[0,t'-D(q)]}\psi$,
$(v.c)$ if $t' < D(q)$, then $\varphi' = \psi'$.

where ψ' is the result of progressing ψ.

(vi) if $\varphi = \chi \mathcal{U}_{[t,t']}\, \psi$, then

$(vi.a)$ if $D(q) \leqslant t$, then $\varphi' = \chi' \wedge (\chi \mathcal{U}_{[t-D(q),t'-D(q)]}\, \psi)$,
$(vi.b)$ if $t < D(q) \leqslant t'$, then $\varphi' = \psi' \vee (\chi' \wedge (\chi \mathcal{U}_{[0,t'-D(q)]}\, \psi))$,
$(vi.c)$ if $t' < D(q)$, then $\varphi' = \psi'$.

where χ' and ψ' are the result of progressing χ and ψ, respectively.

The algorithm can be extended to first order MITL: quantifiers are dealt with in essentially the same way as propositional connectives, *i.e.* by progressing the formula within the quantifiers scope and prepending the quantifier to the result. A problem with this simple approach, however, is that state formulas can no longer be immediately evaluated, if they contain free variables.

5.2 The Model Representation

In this section, we describe the model representation in more detail. Important concerns in the development of this representation are expressivity, compactness, modularity and ease of understanding. If the last is in fact true of the representation we use can be debated, but we believe this to be the case.

5.2.1 States

To express properties of states and static facts, we use a semi-propositional language, consisting of a collection of state variables taking boolean values or values in a finite domain of objects, and a collection of static functions and predicates, including equality, standard arithmetic functions and relations, and propositional connectives. Functions are allowed to take values outside the object domain, for instance in $\mathbb{R}$ or boolean values, so we make no real difference between functions and predicates. We will also be a little sloppy and simply assume that function and predicate arguments are correctly typed, or type checked when evaluated.

Example 5.1 A model for vehicle movement prediction may contain, for instance, objects representing roads and intersections, *e.g.* $\mathtt{r}_{1,2}$, $\mathtt{r}_{2,1}$, $\mathtt{r}_{2,3}$, $\mathtt{r}_{3,3}$, $\mathtt{i}_1$, $\mathtt{i}_2$, *etc.*, and state variables `loc` (the location of the vehicle, a road), `at_start` (T iff the vehicle is at the start of the road) and `moving` (T iff the vehicle is moving). There may also be unobservable state variables, *e.g.* `dst` (the destination of the vehicle, a road).

Static functions may include *e.g.* $\texttt{start}(r)$, $\texttt{end}(r)$ and $\texttt{length}(r)$ (the starting, ending points and length of road r) and $\texttt{distance}(r, r')$ (the distance, or travel time, along the shortest route from r to r'). Other features of the road network, *e.g.* speed limits or the presence of traffic lights at an intersection, are also represented by appropriate functions.

A state of the transition system is an assignment of values to all state variables.

5.2.2 Dynamics

The system dynamics are described by *transition rules* and *state constraints.* For succinctness, transition rules and state constraints may have parameters: the actual set of rules and constraints comprises all possible instances of rule and and constraint schemas.

Transition Rules

A transition rule has the form $\texttt{label} : \gamma \xrightarrow{[\tau,\tau']} \delta$, where γ and δ, the precondition and postcondition of the transition, are state formulas and the interval $[\tau, \tau']$ is the transition time window. The time window bounds τ and τ', may be arbitrary expressions, as long as they evaluate to values in $\mathbb{R}^+$. `label` is the event name associated with the rule.

When the system is in a state satisfying γ, we say the transition rule is *enabled.* When the rule is enabled and the time elapsed since it became enabled is within the rules time window, the system may change state according to the rule. The postcondition is restricted to be a conjunction of equalities between state variables and expressions, interpreted as a list of assignments: the value of the assigned state variable in the new state is the result of evaluating the expression in the state prior to the transition is made.

Example 5.2 Continuing example 5.1, a few possible transition rules describing vehicle movements are:

$$\begin{array}{l}
\texttt{drive} : \\
\quad \texttt{at\_start} \xrightarrow{[\texttt{length(loc)}/180,\infty]} \texttt{at\_start} = \mathrm{F} \wedge \texttt{at\_end} = \mathrm{T}. \\
\texttt{drive\_part} : \\
\quad \texttt{TRUE} \xrightarrow{[0,\infty]} \texttt{at\_end} = \mathrm{T}. \\
\texttt{turn\_to}(r) : \\
\quad \texttt{start}(r) = \texttt{end(loc)} \wedge \texttt{at\_end} \wedge \texttt{moving} \xrightarrow{[2,\infty]} \\
\quad \texttt{loc} = r \wedge \texttt{at\_end} = \mathrm{F} \wedge \texttt{at\_start} = \mathrm{T}.
\end{array}$$

The difference between the `drive` and `drive_part` rules is that the first applies only when the vehicle is at the start of a road segment, and can therefore have a lower time bound depending on the road. In any other case, no lower bound can be assumed.

To model less regular vehicle behaviour, we may add more rules:

$$\begin{array}{l}
\texttt{stop}: \\
\quad \texttt{moving} \xrightarrow{[0,\infty]} \texttt{moving} = \mathrm{F}. \\
\texttt{start}: \\
\quad \neg\texttt{moving} \xrightarrow{[0,\infty]} \texttt{moving} = \mathrm{T}. \\
\texttt{u\_turn}: \\
\quad \texttt{moving} \wedge \neg\texttt{at\_start} \wedge \neg\texttt{at\_end} \xrightarrow{[5,\infty]} \texttt{loc} = \texttt{opposite}(\texttt{loc}).
\end{array}$$

`opposite` is a static function, and may be either given explicitly or defined in terms of `start` and `end`.

State Constraints

A state constraint has the form $\texttt{label} : \gamma : \tau$, where γ is a state formula and τ an arbitrary expression with value in $\mathbb{R}^+$ (the `label` serves no actual purpose).

As with a transition rule, we say that the constraint is enabled when the system is in a state where γ is true. The constraint may not be continuously enabled for a period of time greater than the value of τ, evaluated in the state where the constraint last became enabled, *i.e.* a state constraint may force the system to make a transition within a certain time.

Example 5.3 In our vehicle model, there is really only one constraint: the vehicle can not be moving forever without eventually going somewhere. This may be formulated as

$$\texttt{stay\_in\_road}(r) : \texttt{loc} = r \wedge \texttt{moving} : \texttt{length}(r)/2$$

for each road r.

5.2.3 Expectations and the Expectation Hierarchy

The timed transition system part of the model generates a set of possible developments. Next, the model is enriched with constraints representing expectations on developments, expressed as MITL formulas. Expectations are arranged in a hierarchy, reflecting different degrees of confidence in them.

Expectation MITL

The *Expectation MITL* language consists of the state language, the four MITL temporal operators and a variable binding construct. Arbitrary expressions, with values in $\mathbb{R}^+$, are allowed as interval bounds for the temporal operators.

In section 5.2.1, we described the state language as “semi-propositional”: it is not propositional logic, since reference can be made to object constants, but it is finite and has no quantifiers. This compromise has turned out to be convenient for writing transition rules and state constraints. In MITL formulas, however, quantifiers are commonly used to "carry" values across temporal operators. For example, to state that the value of loc remains the same in the next state, one would typically write $\forall x(\texttt{loc} = x \rightarrow \bigcirc_{[0,\infty]}\texttt{loc} = x)$. Without quantifiers, one would have to write a conjunction covering all possible values for loc.

To enable the writing of compact formulas, but at the same time keep the language computationally simple, we introduce a variable binding operator with the form "$\texttt{let}\, x = \epsilon \,\texttt{in}\, \varphi$", where x is a local variable, ϵ an expression and φ a MITL formula (containing x), and the obvious interpretation. We allow ordinary quantifiers as well, but treat them only as a shorthand for conjunctions or disjunctions over all objects in the domain.

Example 5.4 In our vehicle model, the expectation that the vehicle does not suddenly stop or make a U-turn can be formulated

$$\begin{array}{l} \Box_{[0,\infty]}\texttt{let}\, x = \texttt{loc in} \\ \quad (\texttt{loc} = x \wedge \texttt{moving}\,\mathcal{U}_{[0,\texttt{length(loc)/(spd\_lim(loc)}-5)]}\,\texttt{at\_end}). \end{array} \tag{5.1}$$

The formula states that "it's always the case that the vehicle remains in road x and moving until it is at the end of x, where x is the current location of the vehicle".

Furthermore, the expression $\texttt{length(loc)/(spd\_lim(loc)} - 5)$ sets an upper bound on the time that the vehicle should remain in the road, based on the expectation that it travels at no less than 5 km/h below the speed limit of the road. To express a matching lower bound, a slightly more complicated construction is needed:

$$\begin{array}{l} \Box_{[0,\infty]}\texttt{let}\, x = \texttt{loc in} \\ \quad \bigcirc_{[0,\infty]}(\texttt{loc} = x \vee \\ \qquad\qquad \texttt{let}\, y = \texttt{loc in}\, \Box_{[0,\texttt{length}(y)/(\texttt{spd\_lim}(y)+15)]}\texttt{loc} = y) \end{array} \tag{5.2}$$

This formula states that "(it's always the case that) if x is the current location of the vehicle, then if in the next state the vehicles location is $y \neq x$, the vehicle remains in road y at all times in the interval $[0, \texttt{length}(y)/(\texttt{spd\_lim}(y)+15)]$ from the beginning of the next state".

Another reasonable expectation is that where the vehicle goes depends on where it is going:

$$\begin{array}{l} \Box_{[0,\infty]}\texttt{let}\, x = \texttt{loc in} \\ \quad \Diamond_{[0,\infty]}(\texttt{distance(loc,dst)} < \texttt{distance}(x,\texttt{dst})) \end{array} \tag{5.3}$$

The formula states that the vehicle should eventually move closer to its destination. This is a weak expectation, since it does not rule out that the vehicle makes a detour in between. A stronger form would be

$$\begin{array}{l} \Box_{[0,\infty]}\texttt{let}\, x = \texttt{loc in} \\ \quad \bigcirc_{[0,\infty]}(\texttt{loc} = x \vee (\texttt{distance(loc,dst)} < \texttt{distance}(x,\texttt{dst}))) \end{array} \tag{5.4}$$

stating that whenever the value of loc changes, it has be to a road closer to the vehicles destination.

The Expectation Hierarchy

As stated in the chapter introduction, "a development is normal to the degree that it satisfies expectations". Since a MITL formula can only be true or false, this can only mean that a development is more normal than another iff the set of expectations satisfied by the latter is a strict subset of the set satisfied by the former.

We may, however, have more confidence in some expectations than in others. For example, expectation (5.3) is weaker than (5.4), and hence may be considered more likely. Consequently, we specify a partial order of "inverse confidence" on expectations, and arrange the set of expectations into a hierarchy accordingly.

Definition 5.5 (Relative Normality)
Let $(\Phi, <)$ be a partially ordered set of expectations (where we have more confidence in expectations lower in the hierarchy).

We define a partial order of *relative normality* on developments, in which d is *more normal* than d' iff (*i*) there exists at least one expectation $\varphi \in \Phi$ not satisfied by d', and (*ii*) for every expectation $\varphi \in \Phi$ not satisfied by d, there exists an expectation $\psi \in \Phi$ not satisfied by d' such that $\psi < \varphi$. Note that the relative normality order is strict, *i.e.* non-reflexive.

The set of most normal developments are the maximal elements *w.r.t* relative normality. The set of next most normal developments are the maximal elements among those that are not most normal, and so on. Since the expectation set is finite, only a finite number, say n, of development sets can be distinguished. The *level of normality* of a development is defined by numbering these sets $0, \ldots, n-1$, 0 being the least normal and $n-1$ the most normal, and assigning the development the number of the set it belongs to.

A special case is the linear hierarchy, corresponding to a total order on expectations. In this case, the level of normality of a development is k when it satisfies expectations $1, \ldots, k$, but not expectation $k+1$ (and 0 when it satisfies no expectation).

Example 5.5 As already mentioned, expectation (5.3) is weaker than (5.4), and consequently they should be ordered (5.3) < (5.4). If we believe that people drive too fast more often than too slow, we may also order expectations (5.1) < (5.2). An order between the two sets, however, is difficult to see.

In general, ordering expectations according to confidence is a very important, and often difficult, issue in the construction of a model.

5.3 Prediction Algorithm

Computing predictions to a given time horizon and at a given level of normality breaks down into two parts: (*1*) compute the set of prefixes up to T of developments generated by the timed transition system part of the model[1] and (*2*) determine the normality of each development prefix, *i.e.* which expectations it satisfies.

5.3.1 Generating Finite Developments

Recall that a development consists of a sequence of alternating states and events, $d = q_0, a_0, q_1, a_1, \ldots$, and a function that assigns a starting time to each state (or, equivalently, a time to each event). Rewriting the conditions of definition 5.1 to match the representation described in section 5.2 is trivial.

Even though we only need to compute finite prefixes of developments generated by a timed transition system from a finite set of initial states, the set of possible developments is still uncountable, since the starting time of any state in a development can change by an arbitrarily small amount. To be able to enumerate finite developments, we have to use a more compact representation: a set of developments that differ only on state starting times are represented by a finite sequence of states and events, $d = q_0, a_0, \ldots, q_n$, and a set C of constraints on the starting times $T(q_0), \ldots, T(q_n)$.

It is natural to view the set of developments generated from a starting state q_0 as a tree, where every path constitutes a different development, and we adopt this representation for computing the set. The nodes of the prediction tree are "decorated states", system states annotated with information about enabled transitions and state constraints and the current set of time constraints. Each enabled transition rule gives rise to a successor node, whose state is the result of applying the postcondition of the rule to the state of the parent node.

Time constraints are managed in a *temporal constraint network* (TCN). For a so called *simple TCN*, containing only constraints of the form $X_i - X_j \leqslant c$ or $X_i - X_j < c$ (*i.e.* bounds on the difference between two variables), there exists efficient (polynomial time) algorithms to check consistency and to extract minimal and maximal bounds on the difference between two variables (Dechter, Meiri, & Pearl 1991). For each state q, the variables $X_S(q)$ and $X_E(q)$ denote the starting and ending times of q, respectively.

Enabled transition rules and state constraints are represented by tuples containing the time point at which the rule or constraint became enabled and the actual values of its time bounds. Let $\mathrm{val}(\epsilon, q)$ denote the value of an expression (or state formula) ϵ in state q. In the root node, the sets of enabled rules and constraints contain simply those whose preconditions are true in q_0:

$$\begin{aligned}
\mathcal{R}^E(q_0) &= \{(\gamma \xrightarrow{[\tau,\tau']} \delta, X_S(q_0), v, v') \mid \mathrm{val}(\gamma, q_0) = \mathrm{T}, v = \mathrm{val}(\tau, q_0), \\
&\qquad v' = \mathrm{val}(\tau', q_0)\}, \\
\mathcal{C}^E(q_0) &= \{(\beta : \tau, X_S(q_0), v) \mid \mathrm{val}(\beta, q_0) = \mathrm{T}, v = \mathrm{val}(\tau, q_0)\},
\end{aligned}$$

Time bounds are evaluated in q_0 and also relative to the starting time of q_0. In

[1]Because we ignore executability and non-zenoness properties of the transition system, there is no guarantee that every prefix can be extended into an infinite development.

successor nodes, the calculation is a little more complicated, since the time bounds of already enabled rules and constraints should remain unchanged:

$$\begin{aligned}
\mathcal{R}^E(q_i) &= \{(\gamma \xrightarrow{[\tau,\tau']} \delta, X, v, v') \in \mathcal{R}^E(q_{i-1}) \mid \mathrm{val}(\gamma, q_i) = \mathrm{T}\} \cup \\
&\quad \{(\gamma \xrightarrow{[\tau,\tau']} \delta, X_S(q_i), v, v') \notin \mathcal{R}^E(q_{i-1}) \mid \mathrm{val}(\gamma, q_i) = \mathrm{T}, \\
&\quad\quad v = \mathrm{val}(\tau, q_i), v' = \mathrm{val}(\tau', q_i)\} \\
\mathcal{C}^E(q_i) &= \{(\beta : \tau, X, v) \in \mathcal{C}^E(q_i) \mid \mathrm{val}(\beta, q_i) = \mathrm{T}\} \cup \\
&\quad \{(\beta : \tau, X_S(q_i), v) \notin \mathcal{C}^E(q_i) \mid \mathrm{val}(\beta, q_i) = \mathrm{T}, v = \mathrm{val}(\tau, q_i)\}
\end{aligned}$$

where q_{i-1} is predecessor state of q_i, *i.e.* the state associated with the parent node in the prediction tree.

The set of time constraint is "local" to each node: adding constraints to a node does not affect the parent node, even if the added constraint involves the start or end time of earlier states, nor any other successor of the same parent. Every node has the constraints $X_E(q_i) - X_S(q_i) \geqslant 0$ and $X_E(q_i) - X \leqslant v$ for each $(\beta : \tau, X, v) \in \mathcal{C}^E(q_i)$. A successor node inherits all constraints from its parent, and in addition has the constraints $X_S(q_i) = X_E(q_{i-1})$ and $v \leqslant X_S(q_i) - X \leqslant v'$, where $(\gamma \xrightarrow{[\tau,\tau']} \delta, X, v, v') \in \mathcal{R}^E(q_{i-1})$ is the enabled transition rule by which q_i was created from the parent node q_{i-1}.

5.3.2 Determining Normality

Because the developments we generate are only finite prefixes, expectation MITL formulas may not always have a well defined truth value according to definition 5.3. There are at least two ways in which we can treat this problem: we may assume that no more transitions are made, and thus that the last state of the prefix remains forever, and evaluate the formula in the resulting infinite development, or we may regard a formula as false only if it is provably false in every possible extension of the development prefix. We take the second, more cautious, interpretation.

To prove that an expectation MITL formula can not be satisfied by any extension to a development prefix, we apply progression: if the algorithm returns FALSE, the input formula can not be satisfied by any extension of the development prefix; if it returns TRUE the formula is satisfied by every extension of the prefix; in any other case, the returned formula is a condition to be further progressed through subsequent states.

Algorithm 5.4 assumes that the duration of the input state, $D(q)$, is known exactly, but as explained in section 5.3.1 we have to represent sets of developments by a combination of a state/event sequence and constraints on event times. Therefore, the algorithm has to be extended to take as input a set of time constraints, C, and return the set of *all* possible progressions, $\{(\varphi'_1, C'_1), \ldots, (\varphi'_k, C'_k)\}$, where each C'_i is a set of additional time constraints consistent with C and φ'_i is the result of progressing the input formula φ under constraints $C \cup C'_i$.

The extension is conceptually straightforward, though somewhat complicated in practice: first, notice that for each temporal operator algorithm 5.4 branches depending on the duration of the input state, and that the returned formula is built up according to the recursive path of branches taken. In general, an MITL formula defines a tree structure of possible progressions, with time constraints associated to the branches and resulting formulas at the leaves.

For the formal definition of the progression tree, we need to introduce a special form of each temporal operator: a *relative interval* is written $[X : t, t']$, where X is a TCN variable and $t, t' \in \mathbb{R}^+$, and interpreted as $[X + t, X + t']$. Relative temporal operators are obtained by subscripting any of the standard temporal operators with a relative interval.

Definition 5.6 (Progression Tree)
For an MITL formula φ and a state q with starting and ending times denoted by variables $X_S(q)$ and $X_E(q)$, the *progression tree* $\mathcal{T}_P(\varphi)$ is defined as follows: edges in the tree are labeled with constraints (involving $X_S(q)$, $X_E(q)$, and maybe other TCN variables) and leaf nodes are labeled with formulas.

(*i*) If φ is a state formula (*i.e.* contains no temporal operators), $\mathcal{T}_P(\varphi)$ consists only of a leaf, labeled with TRUE if φ holds in q and FALSE if not.

(*ii*) $\mathcal{T}_P(\neg\alpha)$ is constructed from $\mathcal{T}_P(\alpha)$ by replacing every leaf label α' with $\neg\alpha'$.

(*iii*) To construct $\mathcal{T}_P(\alpha \wedge \beta)$, start with $\mathcal{T}_P(\alpha)$. For every leaf, with the label α', replace the leaf with a copy of $\mathcal{T}_P(\beta)$ and replace every leaf label β' in this copy by $\alpha' \wedge \beta'$.

The construction of $\mathcal{T}_P(\alpha \vee \beta)$ is analogous.

(*iv*) $\mathcal{T}_P(\bigcirc_{[\tau,\tau']}\psi)$ has three branches, labeled $X_E(q) - X_S(q) < \text{val}(\tau, q)$, $X_E(q) - X_S(q) > \text{val}(\tau', q)$ and $\text{val}(\tau, q) \leqslant X_E(q) - X_S(q) \leqslant \text{val}(\tau', q)$ ($\text{val}(\tau, q)$ denotes the value of expression τ in q). At the first two are leaves labeled FALSE, at the last is the root of $\mathcal{T}_P(\psi)$.

(*v*) $\mathcal{T}_P(\Box_{[X:v,v']}\psi)$ has three branches too, labeled $X_E(q) - X < v$, $v \leqslant X_E(q) - X \leqslant v'$ and $v' < X_E(q) - X$. At the first is a leaf labeled $\Box_{[X:v,v']}\psi$. At the second is the root of $\mathcal{T}_P(\psi)$, and in that subtree every leaf label ψ' is replaced with $(\Box_{[X:v,v']}\psi) \wedge \psi'$. At the third is the root of $\mathcal{T}_P(\psi)$, unmodified.

(*vi*) $\mathcal{T}_P(\Box_{[\tau,\tau']}\psi)$ is equal to $\mathcal{T}_P(\Box_{[X_S(q):\text{val}(\tau,q),\text{val}(\tau',q)]}\psi)$.

(*vii*) $\mathcal{T}_P(\Diamond_{[X:v,v']}\psi)$ is similar: it has three branches, labeled $X_E(q) - X < v$, $v \leqslant X_E(q) - X \leqslant v'$ and $v' < X_E(q) - X$. At the first is a leaf labeled $\Diamond_{[X:v,v']}\psi$, at the second $\mathcal{T}_P(\psi)$ with every leaf label ψ' is replaced by $(\Diamond_{[X:v,v']}\psi) \vee \psi'$, and at the third only $\mathcal{T}_P(\psi)$.

(*viii*) $\mathcal{T}_P(\Diamond_{[\tau,\tau']}\psi)$ is equal to $\mathcal{T}_P(\Diamond_{[X_S(q):\text{val}(\tau,q),\text{val}(\tau',q)]}\psi)$.

(*ix*) $\mathcal{T}_P(\chi \mathcal{U}_{[X:v,v']} \psi)$ also has three branches, labeled $X_E(q) - X < v$, $v \leqslant X_E(q) - X \leqslant v'$ and $v' < X_E(q) - X$. At the first is a leaf labeled $\chi \mathcal{U}_{[X:v,v']} \psi$. At the second is $\mathcal{T}_P(\psi)$, but every leaf, with label ψ', is replaced a copy of $\mathcal{T}_P(\chi)$, and in this copy, every leaf label χ' is replaced by $(\chi' \wedge (\chi \mathcal{U}_{[X:v,v']} \psi)) \vee \psi'$. At the third is only $\mathcal{T}_P(\psi)$.

(*x*) $\mathcal{T}_P(\chi \mathcal{U}_{[\tau,\tau']} \psi)$ equals $\mathcal{T}_P(\chi \mathcal{U}_{[X_S(q):\mathrm{val}(\tau,q),\mathrm{val}(\tau',q)]} \psi)$.

(*xi*) $\mathcal{T}_P(\mathtt{let}\, x = \epsilon \;\mathtt{in}\, \psi)$ is $\mathcal{T}_P(\psi)$ in which every leaf label ψ' is replaced by $\mathtt{let}\, x = \mathrm{val}(\epsilon, q) \;\mathtt{in}\, \psi'$.

For every leaf l in the progression tree, the extended algorithm collects the set of constraints C_l found along the path to l: if $C_l \cup C$, where C is the input constraint set, is consistent, (φ'_l, C'_l) is placed in the set of progressions returned. The method of traversing the progression tree is not important: we have used depth-first search, storing current branch choices directly in the formula tree structure so that the progression tree is never explicitly generated.

The solutions returned by the algorithm may contain relative temporal operators, which only make sense in the context of a particular constraint set. Aside from testing if they equal FALSE, however, the only use for the returned formulas is to progress them further through successor nodes, which inherit the constraint set of the parent, so this causes no problem.

Example 5.6 Consider the formula $\varphi = \Box_{[5,9]} \bigcirc_{[0,4]} p$. The progression tree is shown in figure 5.3.2. If the input set of constraints is $C = \{0 < X_E - X_S < 7\}$, there are two consistent solution paths:

(*a*) $X_E - X_S < 5$, with the result $\Box_{[X_S:5,9]} \bigcirc_{[0,4]} p$.

(*b*) $5 \leqslant X_E - X_S \leqslant 9$, $X_E - X_S > 4$, with the result FALSE.

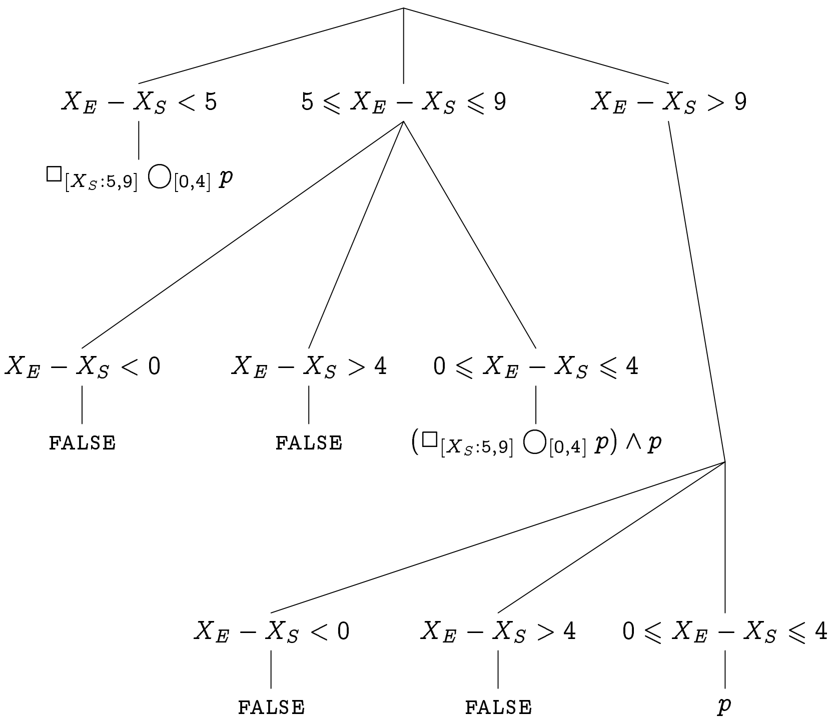
$X_E - X_S > 9$ and $X_E - X_S < 0$ both contradict C, while $0 \leqslant X_E - X_S \leqslant 4$ is inconsistent with the constraint $5 \leqslant X_E - X_S$ labelling the branch above.

5.3.3 The Basic Algorithm

The basic prediction algorithm combines the methods described in sections 5.3.1 and 5.3.2 to compute a set of prediction trees (the *prediction forest*) and determine the normality of every development represented in the forest. To avoid wasting time generating developments of less than desired normality, the two steps are interleaved.

Constructing the Prediction Forest

There is one tree for every possible initial state consistent with known facts. Nodes in each prediction tree are states, decorated as described above, but containing also the



$X_E - X_S < 5$
$5 \leqslant X_E - X_S \leqslant 9$
$X_E - X_S > 9$
$\Box_{[X_S:5,9]} \bigcirc_{[0,4]} p$
$X_E - X_S < 0$
$X_E - X_S > 4$
$0 \leqslant X_E - X_S \leqslant 4$
FALSE
FALSE
$(\Box_{[X_S:5,9]} \bigcirc_{[0,4]} p) \wedge p$
$X_E - X_S < 0$
$X_E - X_S > 4$
$0 \leqslant X_E - X_S \leqslant 4$
FALSE
FALSE
p

Figure 5.1: $\mathcal{T}_P(\Box_{[5,9]} \bigcirc_{[0,4]} p)$

set of expectations on successor states obtained by progressing expectations through the state associated with the node. Expectations on the root nodes are those found in the expectation hierarchy of the model. Since progression may return several results, there may have to be several copies of a node, containing the same state but different constraint sets and successor expectations.

When a node is expanded, each successor state is generated by application of an enabled transition rule and decorated. Expectations stored in the parent node are progressed through the state, which may have to be "split" according to progression results. Nodes with inconsistent constraint sets are discarded, or at least not further expanded.

Enumerating Developments

Every node q such that $X_S(q) \leqslant H \leqslant X_E(q)$ is consistent with the constraints associated with q represents a possible development up to time H. The sequence of states and events in the development is given by the path leading from the root node to q, but the corresponding set of time constraints are those associated with node q, because the constraint set of the parent node is duplicated when a successor is constructed, and only the successor nodes set is modified to reflect that path. The normality of the development equals the normality of the terminal node.

Developments can be enumerated in several ways: the prediction forest can be built breadth first up to the required depth and a set of developments extracted, or terminal nodes can be searched out depth first, avoiding the need to build the forest explicitly.

Which nodes need to be expanded, and when the process stops, depends on the desired horizon and level of normality of the prediction. To make a complete prediction up to a time horizon $H \in \mathbb{R}^+$, every node with an earliest possible ending time (as determined by the constraint set) is less than H must be expanded. For predictions at level k of normality, only nodes with normality k or more have to be expanded, since the normality of a development can never increase.

Integrating Observations

In the prediction problem we are interested in, there are in general few observations, but often more than one. Including observations at a later time than the initial state in the basic algorithm is rather straightforward.

An observation consists of a formula, φ_o, in the state language, observed to be true at time t_o (relative to the time of the initial state). During construction of the prediction forrest, any node such that φ_o is false in the associated state q is split into two copies, and the constraints $X_E(q) < t_o$ and $t_o < X_S(q)$ are added to the constraint set of each copy respectively.

Observations are important, since they may allow us to infer facts about unobservable state variables. For example, observations of a sequence of route choices made by the vehicle, combined with an expectation such as (5.4), may make it possible to rule out some vehicle destinations, whereas with only observation, there would be a complete prediction tree for each.

5.4 Query Answering

The prediction forrest is too large and wild-grown to be a useful decision aid in itself: we need to fell from it manageable pieces of information. A way to do this is by asking *queries* about the future, and computing answers based on predicted developments.

5.4.1 Boolean Queries

To begin with, consider a simple boolean query, *e.g.* "is the vehicle in road $\mathtt{r}_4$ at some point in $[2.0, 2.4]$?". This kind of query is naturally expressed as an MITL formula, $\Diamond_{[2.0,2.4]}\mathtt{loc} = \mathtt{r}_4$. For the formula to be evaluable, it must be specified which development it refers to. We distinguish only two modes of querying: the query is *possibly* true iff it is true in some development, and *necessarily* true iff it is true in every development. In addition, a query may be made with respect to a level of normality, specifying which developments should be considered in computing the answer.

To specify the mode of a query, we introduce two modal operators. They are essentially equivalents of the path quantifiers employed in the temporal logics CTL and CTL* (Emerson 1990).

Definition 5.7 (Query Formula)
A *query formula* has the form $P^k\varphi$ or $N^k\varphi$, where $k \in \mathbb{N}$ and φ is an MITL formula.

The intended meaning of P^k is "possibly, at level k of normality", while N^k is the corresponding necessity modality.

The answer to a query formula is T or F, in some cases complemented by a so called *witness*: for a possibility query, a witness exists if the answer to the query is T, and is a development, normal at the specified level, satisfying the querys MITL formula; for a necessity query, the witness is a development violating the MITL formula, which exists if the answer is F.

The extended progression algorithm could be used to evaluate queries in each development, but this would be quite complicated and probably inefficient. Instead, we develop a specialized evaluation algorithm for a restricted, but still useful, class of formulas.

Algorithm 5.8 (Query Evaluation)
Let φ be a query formula of the form $N^k\Box_{[t,t']}\psi$, $N^k\Diamond_{[t,t']}\psi$, $P^k\Box_{[t,t']}\psi$ or $P^k\Diamond_{[t,t']}\psi$, where $t, t' \in \mathbb{R}^+$ and ψ is a state formula.

Furthermore, let $\mathcal{D}^k_H = \{(d_0, C_0), (d_1, C_1), \ldots\}$ denote an enumeration of the set of developments normal at level k or above up to time H (in (d_i, C_i), d_i denotes the state/event sequence and C_i the set of constraints).

(i) If $\varphi = N^k\Box_{[t,t']}\psi$, then for every $(d, C) \in \mathcal{D}^k_{t'}$ and for every state $q \in d$ such that at least one of

(i.a) $X_S(q) \leqslant t < X_E(q)$

(i.b) $X_S(q) \leqslant t' < X_E(q)$

(i.c) $\{t \leqslant X_S(q), X_E(q) \leqslant t'\}$

is consistent with C, if ψ does not hold in q return F with (d, C) as witness. If $\mathcal{D}^k_{t'}$ is exhausted and no counterexample found, return T.

(ii) If $\varphi = N\Diamond_{[t,t']}\psi$, then for every $(d, C) \in \mathcal{D}^k_{t'}$, find a subsequence $q_i, \ldots, q_j$ of d such that ψ holds in each state in the subsequence and such that $t' < X_S(q_i)$ and $X_E(q_j) \leqslant t$ are each (individually) inconsistent with C (*i.e.* $q_i, \ldots, q_j$ necessarily overlaps with the interval $[t, t']$). If no such subsequence exists, return F with (d, C) as witness. If one is found in every development, return T.

(iii) If $\varphi = P\Box_{[t,t']}\psi$ then for every $(d, C) \in \mathcal{D}^k_{t'}$, find a subsequence $q_i, \ldots, q_j$ of d such that ψ holds in each state in the subsequence and such that $\{X_S(q_i) \leqslant t, t' \leqslant X_E(q_j)\}$ is consistent with C (*i.e.* $q_i, \ldots, q_j$ possibly contains the interval $[t, t']$). If such a subsequence exists, return T with (d, C) as witness. If none is found in any development, return F.

(iv) If $\varphi = P\Diamond_{[t,t']}\psi$, find a $(d, C) \in \mathcal{D}^k_{t'}$ and a state $q \in d$ such that at least one of

(iv.a) $X_S(q) \leqslant t < X_E(q)$

(iv.b) $X_S(q) \leqslant t' < X_E(q)$

(iv.c) $\{t \leqslant X_S(q), X_E(q) \leqslant t'\}$

is consistent with C. If such a development and state can be found, return T with (d, C) as witness, otherwise return F.

Algorithm 5.8 makes no commitment to any specific enumeration order among developments, and does not require the set $\mathcal{D}^k_{t'}$ to be explicitly constructed. Concievably, heuristics could be developed to focus the search for an example/counterexample, although in the worst case every development has to be examined.

5.4.2 Value Queries

For the reidentification problem in scenario 1, restricted query formulas are sufficient since we need only to ask if a particular situation is likely. In the search problem in scenario 2, however, we need to find a set of vehicle locations and corresponding time intervals that are likely. To find the set of all values of an expression that occur in every development, or in at least one, is a straightforward excercise in tree traversal, but we are interested in values that occur at the same time, or nearly at the same time, in every development.

Definition 5.9 (Point and Interval Value Set)
Let ϵ be an expression in the state language.

The *point value set* for ϵ, *w.r.t.* normality k and horizon H, consists of all pairs $([t, t'], v)$, $t, t' \leqslant H$, such that $N^k \Diamond_{[t,t']} \epsilon = v$ holds and such that $[t, t']$ is minimal, in the sense that $N^k \Diamond_{[t'',t''']} \epsilon = v$ does not hold for any interval $[t'', t''']$ strictly contained in $[t, t']$.

The *interval value set* for ϵ, *w.r.t.* normality k and horizon H, consists of all pairs $([t, t'], v)$, $t, t' \leqslant H$, such that $N^k \Box_{[t,t']} \epsilon = v$ holds and such that $[t, t']$ is maximal, in the sense that $N^k \Box_{[t'',t''']} \epsilon = v$ does not hold for any interval $[t'', t''']$ containing $[t, t']$ strictly.

Thus, the point value set consists of the minimal interval/value pairs such that the expression ϵ necessarily assumes the value at some point in the interval, while the interval value set consists of the maximal interval/value pairs such that the expression ϵ necessarily assumes the value throughout the entire interval.

Although a generate-and-test procedure based on algorithm 5.8 could be used to find point and interval value sets, it is possible to do somewhat better.

Algorithm 5.10 (Value Set Extraction)
Let ϵ be an expression in the state language, and let $\mathcal{D}_H^k = \{(d_0, C_0), (d_0, C_0), \ldots\}$ be an enumeration of the set of developments normal at level k or above up to time H.

Denote the point value set by V_P and the interval value set V_I (to avoid excessive notational clutter, the fact that they are value sets for ϵ *w.r.t.* normality k and horizon H is implicit throughout the description of the algorithm). The procedures extracting V_P and V_I are very similar: both extract from $(d_0, C_0) \in \mathcal{D}_H^k$ an initial set and then filter this set with each of the remaining developments in turn.

For any development (d, C), a set of values $V(\epsilon, (d, C))$ is computed by "segmenting" d into maximal subsequences of states $q_i, \ldots, q_j$ such that ϵ has value v in every state in the subsequence. For every such subsequence $q_i, \ldots, q_j$, $([t, t'], v) \in V(\epsilon, (d, C))$, where t is the latest starting time of q_i (*i.e.* the maximal value of $X_S(q_i)$ consistent with C) and t' the earliest ending time of q_j (if $t' < t$, the pair is omitted).

The initial point and interval value sets, V_P^0 and V_I^0, both equal $V(\epsilon, (d_0, C_0)$. For each remaining development (d_i, C_i), V_P^i and V_I^i are computed from V_P^{i-1} and V_I^{i-1}, respectively, together with $V(\epsilon, (d_i, C_i))$: For every $([t, t'], v) \in V_P^{i-1}$, find every $([t'', t'''], v) \in V(\epsilon, (d_i, C_i))$ and form a value pair with the union of the intervals, $([\min(t, t''), \max(t', t''')], v)$. V_P^i consists of those value pairs that are minimal *w.r.t* interval inclusion. For every $([t, t'], v) \in V_I^{i-1}$, find every $([t'', t'''], v) \in V(\epsilon, (d_i, C_i))$

and form a pair with the intersection of the intervals, $([\max(t, t''), \min(t', t''')], v)$ (if $\min(t', t''') < \max(t, t'')$, the pair is omitted). V_I^i consists of those value pairs that are maximal *w.r.t* interval inclusion.

After all developments have been processed, the sets V_P^n resp. V_I^n remaining are returned.

The algorithm is complete *w.r.t.* interval value sets, but not for point value sets. The reason for this is that the set $V(\epsilon, (d, C))$ computed for each development is in fact an interval value set, *i.e.* for every value pair $([t, t'], v) \in V(\epsilon, (d, C))$, $[t, t']$ is a maximal interval that such v necessarily holds throughout $[t, t']$ in that development, and therefore may miss some valid point sets. It is not difficult to construct an example of a development such that $V(\epsilon, (d, C)) = \emptyset$, but a point set always exists.

5.5 Evaluation

In chapter 3, we suggested three points to consider when evaluating a model design: representation, computation and integration with the application. We discuss each point in turn, the first two with regard to expectation/normality models in general, and the last in the context of the problem and scenarios described in chapter 4.

In chapter 7, we contrast our two model designs, and present some preliminary experimental results to support assessments.

5.5.1 Representation

The timed transition system formalism and expectation MITL are powerful, and likely sufficient to express dynamics and expectations in any world we may want to consider. The representation of expectation/normality models by timed transition systems and MITL is clearly most suited to model acquisition by encoding, which means that besides expressivity, characteristics such as modularity, transparency and "naturalness" of expression are important. We have also strived for them in developing the representation.

A potential weakness of the representation for expectations is that it lacks a mechanism for expressing exceptions modularly. Consider, for example expectation (5.1), which states that a vehicle normally keeps moving: a reasonable qualification to this expectation is that the vehicle does not stop unless it has a good reason, *e.g.* a red light at the next intersection. A solution is to rewrite the expectation, *e.g.* as

$$\begin{aligned}
&\Box_{[0,\infty]}\mathtt{let}\, x = \mathtt{loc}\ \mathtt{in} \\
&\quad ((\mathtt{loc} = x \wedge \\
&\qquad (\neg\mathtt{moving} \rightarrow (\mathtt{has\_light}(\mathtt{loc}) \wedge \Diamond_{[0,30]}\Box_{[0,30]}(\mathtt{moving} \vee (\mathtt{loc} \neq x))))) \\
&\quad \mathcal{U}_{[0,\mathtt{length}(\mathtt{loc})/(\mathtt{spd\_lim}(\mathtt{loc})-5)]}\ \mathtt{at\_end}).
\end{aligned} \tag{5.5}$$

stating that if the vehicle stops, then there should be a traffic light at the end of the road section it is currently in, and it should soon start moving and while it remains in the same road should not stop again for at least the time that a traffic light stays green once it has changed (in this case 30 seconds). The example illustrates that while exceptions can be encoded, it may be difficult to do so in a modular way.

The advantages and disadvantages of normality contra probability as a measure of relative likelihood can probably be debated at length. We note only that normality, being a coarser measure, obviously fails to take advantage of statistical estimates of probability, when such are available. It is, on the other hand, better able to express the situation when we have, and know that we have, not enough information to distinguish the relative likelihood of two developments.

Another point to note is that while deciding if a development satisfies a given MITL formula is a logical operation, the assessment of normality is not a form of logical inference: the choice of expectations and their arrangement into a hierarchy is arbitrary, and the normality measure has in a sense more in common with "prototype based" representations, such as frames (Hayes 1979). As shown in example 5.4, many expectations, although not written as implications, carry an "if-then" meaning: "in this situation, the normal course of action would be ...". It is entirely possible that some "schema based" representation, *e.g.* the chronicles described by Ghallab (1996), would be better suited to express expectations.

Learning Expectations

Although the expectation/normality model representation is primarily geared towards an encoding mode of model acquisition, the possibility of learning expectations from examples of normal and exceptional developments is interesting to consider.

There is, as far as we are aware, no work on learning MITL formulas from examples. However, MITL and timed transition systems are "equivalent", in the sense that it is possible to construct a timed transition system that accepts the same set of developments, for a restricted class of MITL formulas (Alur, Feder, & Henzinger 1996), and there is a substantial amount of work on inferring the structure of an (untimed) deterministic finite automaton (DFA). Pitt (1989) provides a survey.

Learning a DFA from examples positive and negative examples of the language accepted by the automaton is known to be equivalent to a number of hard number theoretic problems: inverting the RSA encryption function, factoring Blum integers and deciding quadratic residues (Kearns & Valiant 1989). If, however, the learner is allowed to "experiment", asking questions about the membership of a specific string in the language of the automaton and about the correctness of a hypothesis (equivalence questions), a DFA can be identified in polynomial time (Angluin 1987). Whether any of these results extend to timed automata is unknown.

Prediction and Planning

As mentioned in section 2.4, an essential part of planning is to predict the result of executing a candidate plan. Although no special considerations have been made for plans in the model design, they can fairly easily be expressed in terms of transition rules and state constraints. The interaction between the plan and its environment is modelled using state variables: actions available to the plan executor take the form of transition rules and affect a set of variables, which in turn influence transitions made by the environment. A different set of variables, affected by environment transitions, represent sensory input to the plan executor, and state constraints force the next plan action to happen within the executors response time.

5.5.2 Computation

The prediction algorithm is inherently exponential: since there are in most states more than one applicable transition rule, the set of possible developments, and hence the prediction forest, grows in general exponentially with depth, *i.e.* prediction horizon. Strong expectations, however, may limit uncertanity and if strong enough may bring the branching factor close to 1, resulting in almost linear growth. Another source of intractability is in the set of initial states, which grows exponentially with the number state variables whose initial values are unknown.

Improvements to the Basic Algorithm

A way to counter the exponential growth of the initial state set would be an increased use of constraints, allowing state variables, like time variables, to have only constrained values. The disadvantage is that it is then no longer possible to directly evaluate state formulas: evaluation needs to be replaced by a mechanism similar to the extended progression algorithm, *i.e.* one taking as input a state formula γ and a partially specified state, and returning one or more sets of additional constraints necessary for γ to be true. How this extension to the progression algorithm is made for propositional MITL is shown in section 5.6.3, but progression and evaluation over partial states for the complete state language is far more complicated.

If the predicted system consists of several independently acting agents, *e.g.* several vehicles, the prediction forest contains, unnecessarily, every possible interleaving of transitions made by each agent. To avoid this, we may compute prediction forests for each agent separately, and combine them only when needed, *e.g.* to answer a query such as "is it possible for vehicles c_1 and c_2 to occupy the same road at some point in $[3.5, 4.0]$". Similar techniques, based on factored representations, have been used to improve efficiency in model checking (Godefroid & Wolper 1993) and in computing approximate MDP solutions (Boutilier, Dean, & Hanks 1999).

5.5.3 Integration

It is not difficult to see that the *form* of predictions provided by the query methods matches the requirements outlined in the UAV task scenarios: reidentification requires only a simple query, while point or interval value sets can be used as input to search planning. A possible problem arises if a plan can not be made to cover all the indicated locations, since the coarseness of the normality measure means that it gives little guidance in choosing which locations to disregard.

Whether the *accuracy* of predictions made using an expectation/normality model can be made good enough can of course only be determined through a process of trial-and-error.

5.6 Relation to Formal Verification Techniques

The model design developed in this chapter borrows heavily from the area of formal hardware/software verification, in particular from the approach called model checking. Therefore, we conclude this chapter by noting some points of relation between the techniques we have developed and those found in that area.

5.6.1 Model Checking

In the model checking approach to verification, the problem is viewed as proving that a model of the actual system satisfies a formal specification of the intended system. The model takes the form of a transition system and the specification is written in some temporal logic. Clarke *et al.* (1999) provide a comprehensive introduction to model checking.

Model checking for real time systems uses timed transition systems and metric temporal logics. There are algorithms for checking specifications in two logics: MITL (Alur, Feder, & Henzinger 1996) and TCTL (Alur, Courcoubetis, & Dill 1993).

Model Checking MITL

The model checking problem for MITL is decidable only under certain restrictions: there is no *next* operator, and operator time intervals are required to be *non-singular*, *i.e.* not consisting of a single point, and to have rational constant bounds, although they may be open or closed at either end, and may be unbounded (Alur, Feder, & Henzinger 1996).

The model checking algorithm for MITL is based on constructing a timed transition system that accepts exactly the developments in which the formula holds. A system is constructed for the negation of the specification formula and composed with the transition system modelling the system to be verified. The composite system is then checked for emptiness, *i.e.* whether there exists at least one development accepted by that system. If no such development exists, every development of the system model satisfies the specification formula (since it fails to satisfy its negation).

Model Checking TCTL

Formulas in Computation Tree Logic (CTL) (Emerson 1990; Clarke, Emerson, & Sistla 1986) combine each temporal operator with a path quantifier, essentially equivalent to the necessity/possibility modal operators introduced in section 5.4. Thus, CTL formulas are not evaluated relative to a development, but to a starting state. For example, the formula $\forall\Diamond\psi$ holds in a state q iff ψ holds in some future state of every development starting in q, while $\exists\Box\psi$ holds in q iff there is at least one development starting in q in which ψ holds in every state.

Timed CTL (TCTL) (Alur, Courcoubetis, & Dill 1993) has no *next* operator, but extends remaining temporal operators are with time constraints, in the form of single equality or inequality to a rational constant. For example, $\forall\Diamond_{=4}\psi$ holds in a state iff ψ holds exactly 4 time units later, along every development. The model checking algorithm for TCTL is based on labeling states of the transition system with the subformulas of the specification formula that hold in each state, starting from the atomic parts of the formula and working "upwards".

5.6.2 Prediction by Model Checking

Although the aim of model checking is different from ours, and the timed transition system representation and temporal logics used are more restricted, model checking techniques could concievably be applied to the problem of prediction.

Consider a predictive model, consisting of a timed transition system S and a linear expectation hierarchy $\varphi_1, \ldots, \varphi_n$, and a query formula of the form $N^n\psi$. Evaluation of the query may be cast as a model checking problem: "does S satisfy $\varphi_1 \wedge \ldots \wedge \varphi_k \rightarrow \psi$?", and if the formulas $\varphi_1, \ldots, \varphi_n$ and ψ meet the necessary restrictions, this question is decidable by the procedure for model checking MITL.

For a query formula of the form $P\psi$, the translation is not as easy. The restricted query formulas handled by algorithm 5.8 are TCTL formulas, but expectation formulas generally are not. If no expectation formulas contains nested temporal operators or *next* operators, they may be expressed in TCTL by prefixing every modal operator with the $\forall$ path quantifier.

Although it may thus be possible to reduce prediction to model checking, it does not appear practical: it severely restricts the expressive power of expectations, and model checking algorithms for both MITL and TCTL require time proportional to, among other things, the size of the specification formula.

5.6.3 MITL Satisfiability

The construction by Alur *et al.* (1996), which is the basis for the MITL model checking algorithm described in section 5.6.1, can be used also to decide the satisfiability of a propositional MITL formula φ, assuming φ meets the restrictions, by constructing a timed transition system accepting the same developments as φ and checking it for non-emptiness.

Based on the extended progression algorithm, we sketch an alternative decision procedure for MITL satisfiability. It is essentially a tableaux algorithm, somewhat similar to decision procedures for the non-real time temporal logic LTL (Gerth *et al.* 1995; Schwendimann 1998). The restriction to non-singular intervals can not be relaxed, since satisfiability of MITL formulas with singular intervals is provably undecidable (Alur & Henzinger 1994). Neither can the asymptotic complexity be improved over that of the previously known algorithm, since the problem is known to be EXPSPACE-complete (Alur, Feder, & Henzinger 1996). The algorithm sketched below is not complete, and is still under development: a preliminary report may be found in (Haslum 2002).

First, suppose the progression algorithm accepts as input a state q with unspecified values for all propositions, as well as unspecified starting and ending time (except for the constraint that $D(q) > 0$), and returns all possible combinations of progressions and constraints, both time constraints and constraints on state values. This can be achieved by a small change to the progression tree:

(*i'*) For a single proposition p, $\mathcal{T}_P(p)$ has two branches, labeled by $p = \mathrm{T}$ and $p = \mathrm{F}$, ending in leafs labeled TRUE and FALSE, respectively.

Note that the branch labels are constraints on the value of p, while the leaf labels are formula constants. Second, progress every returned formula through another unspecified state with the constraints returned by the first progression as input, and every result of this through yet another unspecified state, and so on. If some sequence of progressions eventually results in TRUE, then φ is satisfiable: any development satisfying the set of constraints collected along the way is a model of the formula. If, on the other hand, every sequence of progressions eventually ends in FALSE, then φ is not satisfiable.

Incompleteness stems from the fact that some formulas generate infinite sequences of progressions, *e.g.* $\Box_{[0,\infty]}\Diamond_{[0,10]}p$ or $(\Box_{[0,\infty]}\Diamond_{[0,\infty]}p) \wedge (\Diamond_{[0,\infty]}\Box_{[0,\infty]}\neg p)$. An infinite sequence of progressions corresponds to an infinite cyclic development, and thus if a cycle is detected, the formula can be evaluated directly in the corresponding development. The problem that remains is how to detect all cycles.

Chapter 6

Markov Models

In this chapter, we develop our second model design, based on a Markov process. The relative likelihood of different developments is represented by probabilities, to enable the best possible use of whatever statistical knowledge we may posess.

Like expectation/normality models, it has discrete states at a high level of abstraction. Conceptually, model dynamics are event driven, but for computational reasons we use a "soft discretization" of time, making it in practice more of a step-based representation.

6.1 Technical Background

This section briefly introduces discrete and continuous time Markov processes and the phase method. For a more detailed treatment, see *e.g.* Tijms (1994) or Hoel *et al.* (1972).

6.1.1 Markov Chains

A Markov process is a stochastic transition system, *i.e.* one in which transitions are chosen at random according to some probability distribution. The distinguishing property of Markov processes is that the distribution over transitions depends only on the present state.

A discrete time Markov process is often called a Markov *chain*.

Definition 6.1 (Markov Chain)
A *Markov chain* formally consists of a finite or countably infinite set of *states* $\mathcal{X}$ and a sequence of random variables $X_0, X_1, \ldots$ taking values in $\mathcal{X}$. The value of X_i is said to be the state of the process at *stage* i. The distribution of X_i, for all $i \geqslant 1$,

is such that it satisfies the Markov property:

$$\begin{aligned}&\mathbf{P}(X_i = x_i \mid X_0 = x_0, \ldots, X_{i-1} = x_{i-1}) = \\ &\quad \mathbf{P}(X_i = x_i \mid X_{i-1} = x_{i-1})\end{aligned} \tag{6.1}$$

i.e. the probability of a certain state x materializing at stage i depends only on the state of the process at stage $i-1$, and not on its previous history. The probabilities $\mathbf{P}(X_i = x_i \mid X_{i-1} = x_{i-1})$ are called the *transition probabilities.* If

$$\mathbf{P}(X_i = x \mid X_{i-1} = x') = \mathbf{P}(X_j = x \mid X_{j-1} = x') \tag{6.2}$$

for all $i, j \geqslant 1$, *i.e.* the transition probabilities are independent of time, the chain is said to be *stationary.*

For the remainder of this chapter, we consider only stationary chains and therefore introduce the abbreviations

$$\begin{aligned}P(x,x') &= \mathbf{P}(X_{i+1} = x' \mid X_i = x) \\ P^n(x,x') &= \mathbf{P}(X_{i+n} = x' \mid X_i = x)\end{aligned}$$

Let $\pi_i(x) = \mathbf{P}(X_i = x)$, $x \in \mathcal{X}$, denote the distribution of a Markov chain at stage i. Because of stationarity and the Markov property, π_i and the transition probabilities determine the distribution at every subsequent stage:

$$\pi_{i+k}(x) = \sum_{x' \in \mathcal{X}} \pi_i(x') P^k(x', x) \tag{6.3}$$

A stationary Markov chain is therefore characterized by the transition probabilities and the initial distribution π_0.

6.1.2 The Markov Jump Process

It is straightforward to think of the transitions between each stage of a Markov chain as events, and from there to introduce a delay of random length inbetween each event. For the resulting continuous time stochastic process to be Markovian, however, certain conditions have to be placed on the event time distribution.

Let $X(t) \in \mathcal{X}$, $t \geqslant 0$ denote the state of the process at *time* t. The sequence $X(t_0), X(t_1), \ldots$ obtained by "sampling" at times $t_0 < t_1 < \ldots$ just after every state change of the process is called the *embedded chain* of the process.

We consider only a particular kind of Markov process, the Markov jump process, which has nice computational properties.

Definition 6.2 (Markov Jump Process)
A Markov jump process is a stochastic process $X(t)$, $t \geqslant 0$ such that the embedded chain is a stationary Markov chain and such that the time in each stage satisfies the following conditions:

(i) the time is exponentially distributed with a mean $1/\lambda(x)$ that depends only on the state of the process at the stage ($\lambda(x)$ is called the *leaving rate* of state x),

(ii) the number of stages that occur in a finite time interval is finite with probability 1.

Condition (ii) serves to exclude some troublesome pathological cases. By convention, the transition probability $P(x, x) = 0$ for all $x \in \mathcal{X}$, since this ensures that the time in a stage is unambigously defined.

If the leaving rate is $\lambda(x) = \lambda$ for all $x \in \mathcal{X}$, the process is said to be *uniform*. The process is stationary iff the embedded chain is.

The leaving rates and the transition probabilities do not in the general case uniquely determine a continuous time Markov jump process. However, when the state space is finite, the process is unique and a distribution $\pi_t(x) = \mathbf{P}(X(t) = x)$, $x \in \mathcal{X}$, determines the distribution for all $t' > t$. If the process is stationary and uniform with leaving rate λ, this distribution is

$$\pi_{t+\Delta}(x) = \sum_{x' \in \mathcal{X}} \pi_t(x') \sum_{k \geqslant 0} F_P(k, \lambda\Delta) P^k(x', x) \tag{6.4}$$

where $F_P(k, \mu)$ denotes the Poisson probability density function with mean μ, defined as

$$F_P(k, \mu) = e^{-\mu} \frac{\mu^k}{k!}. \tag{6.5}$$

$F_P(k, \lambda\Delta)$ may be interpreted as the probability of seeing k events in Δ units of time, when the time between any two events is exponentially distributed with intensity λ, *i.e.* events happen "on average" once every $\frac{1}{\lambda}$ time unit.

6.1.3 The Phase Method

The main limitation of the Markov jump process, from the point of view of our application, is the restriction to exponentially distributed stage durations. To circumvent it, we make use of an "encoding trick" known as the phase method, invented by A. K. Erlang in the early 20th century. The foundation of the method is the following theorem.

Theorem 1
Let $F(x)$ be any non-negative distribution and for any fixed $\delta > 0$ let

$$F_\delta(x) = \sum_{k=1}^{\infty} p_\delta(k) \left(1 - \sum_{i=0}^{k-1} e^{-\frac{1}{\delta}x} \frac{(\frac{1}{\delta}x)^k}{k!} \right) \tag{6.6}$$

where

$$p_\delta(k) = F(k\delta) - F((k-1)\delta) \tag{6.7}$$

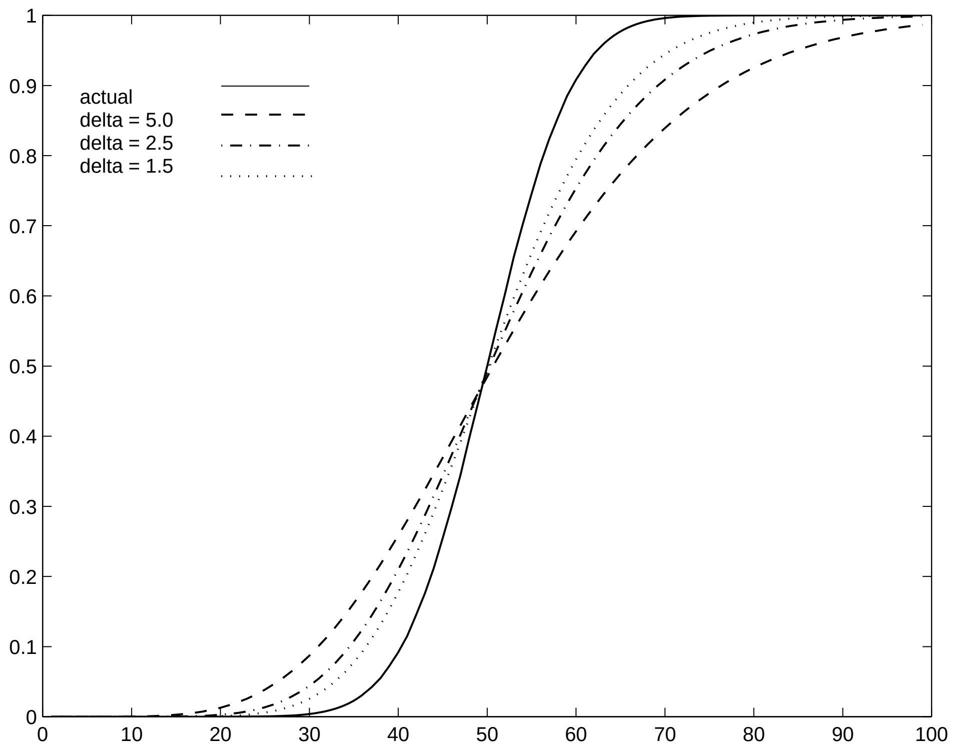


Figure 6.1: Approximation of a normally distributed variable by phases.

for $k = 1, 2, \ldots$. Then $\lim_{\delta \to 0} F_\delta(x) = F(x)$.

The theorem states that any non-negative distribution can be approximated with arbitrary precision by a sum of exponentially distributed variables all with the same rate parameter, $1/\delta$. The name of the method derives from viewing the variable with distribution $F(x)$ as consisting of a number of consecutive phases, each of which is exponentially distributed.

Figure 6.1 shows the distribution function of a normally distributed variable ($\mu = 50.0$, $\sigma = 7.5$) and its approximation by the phase method for different values of δ (5.0, 2.5 and 1.5). As evidenced by the curves, the approximate distribution function has the correct shape, but tends to have a greater deviation (17.5, 13.5 and 11.5, respectively).

Through use of the phase method, any continous time Markov process can be approximated by one which is a uniform Markov jump process. A state in the new process consists of a state in the original process and a counter, *i.e.* $\mathcal{X}' = \mathcal{X} \times \mathbb{N}$.

Transition probabilities may be defined in several ways, for example

$$\begin{array}{lcll} P'((x,1),(x',k)) & = & P(x,x')p_{F_x,\delta}(k) & k=1,2,\ldots \\ P'((x,k),(x,k-1)) & = & 1 & k>1, \end{array} \tag{6.8}$$

where $P(x,x')$ and F_x are the transition probabilities and event time distributions, respectively, of the original process and $p_{F_x,\delta}(k)$ is defined according to equation (6.7) for distribution F_x and a fixed δ. Thus, whenever the original process would make a transition from state x to x', the new process makes the same transition and chooses the number of phases to remain in x'. The event time distribution of the new process is exponential with intensity $1/\delta$ for every state.

Alternatively, we may define $r_{F_x,\delta}(1) = p_{F_x,\delta}(1)$ and $r_{F_x,\delta}(k)$ recursively through

$$r_{F_x,\delta}(k) = \frac{p_{F_x,\delta}(k)}{\prod_{i=1}^{k-1}(1 - r_{F_x,\delta}(i))}, \tag{6.9}$$

and then define the transition probabilites of the new process as

$$\begin{array}{lcl} P'((x,k),(x',1)) & = & P(x,x')r_{F_x,\delta}(k) \\ P'((x,k),(x,k+1)) & = & 1 - r_{F_x,\delta}(k). \end{array} \tag{6.10}$$

In this case, the process chooses after having been in state x for k phases with probability $r_{F_x,\delta}(k)$ to change state, according to the transition probabilities of the original process, and reset the phase counter, or with probability $1 - r_{F_x,\delta}(k)$ to remain in the same state for one more phase. By the definition of $r_{F_x,\delta}(k)$, the probability of the process remaining in the same state for k phases becomes $p_\delta(k)$, for any k.

6.2 Prediction and Estimation

The predictive model is represented by a stationary Markov process, with a finite state space. When the system state at some time t_0 is known, prediction is quite straightforward, but in most applications, *e.g.* prediction of vehicle movements, the system state is not completely observable, and therefore the main part of the problem lies in estimating the initial probability distribution over model states from one or more observations and knowledge of *a priori* distributions.

We describe first the prediction procedure for the case of a given initial process distribution, then a general estimation procedure for the initial process distribution. The estimation procedure has a certain similarity to the predictive filter described in section 2.1.

Algorithm 6.3 (Prediction)
Let $\mathcal{X}$, $P(x \in \mathcal{X}, x' \in \mathcal{X})$ and $F_{x\in\mathcal{X}}(t)$ be the state space ($\mathcal{X}$ is finite), transition probabilities and family of event time distributions describing a stationary continuous

time Markov process, let π_{t_0} be the process distribution at time t_0, and let h be the prediction horizon.

First, the process is transformed, using the technique described in section 6.1 into a finite[1] stationary uniform Markov jump process. Then, the process distribution at $t_0 + h$ is computed through equation (6.4).

Algorithm 6.4 (Estimation)
Let the process be as in algorithm 6.3, let obs(x) denote the observable part of a state x, and let (o_i, t_i), $i = 0, \ldots, n$ be a series of observations (with time stamps).

The relative likelihood of the initial state being x given the observations is, according to Bayes rule,

$$\mathbf{P}(X(t_0) = x) \prod_{i=1}^{n} \mathbf{P}(\text{obs}(X(t_i)) = o_i \mid X(t_0) = x) \tag{6.11}$$

where $\mathbf{P}(X(t_0) = x)$ is the *a priori* probability of the initial state being x. The probability of an observation conditioned on the initial state, $\mathbf{P}(\text{obs}(X(t_i)) = o_i \mid X(t_0) = x)$, is calculated as

$$\sum_{\{x' \in \mathcal{X} \mid \text{obs}(x') = o_i\}} \pi_{t_i}^{x}(x') \tag{6.12}$$

where $\pi_{t_i}^{x}$ is the process distribution at time t_i, computed by algorithm 6.3 for an initial distribution $\pi_{t_0}^{x}$ concentrated to state x, *i.e.*

$$\pi_{t_0}^{x}(x') = \begin{cases} 1 & \text{if } x' = x \\ 0 & \text{elsewhere} \end{cases} \tag{6.13}$$

Equation (6.11) gives relative likelihood: the most likely initial distribution $\pi_{t_0}^{*}$ is obtained by normalizing. Note that the distribution is over the state of the process at the time of the *first* observation.

As an alternative to algorithm 6.4, estimation and prediction may be treated as separate problems, and the most likely process distribution at the time of the *last* observation calculated, based on *a priori* knowledge and observations. There are several advantages to this approach: the prediction horizon is shortened, there is a choice of estimation methods, and finally, not only the initial process distribution but also other parameters of the process can be estimated. For example, observations can be used to adjust the event time distributions of the process, something that can not be done using only algorithm 6.4 unless the process state space is made infinite.

[1]The phase counter introduced into the process state by the transformation is unbounded, so technically, the state space is no longer finite. However, most reasonable event time distributions, *e.g.* the normal distribution, result in a probability $p_\delta(k)$ that is negligably close to zero for all but a finite range of k.

6.3 Model Constructions

The main motivation for representing the predictive model by a Markov process is to make the most use of available traffic statistics. Thus, the transition probabilites and event time distribution of this process should, as far as possible, be based on such information.

The kinds of traffic statistics and planning models we may have at our disposal to construct a model from were discussed in chapter 4, but let us briefly summarize and recall some notation: let $\mathcal{N}$ and $\mathcal{L}$ denote the set of nodes and links in the network, respectively, and $\mathcal{R}_{mn}$ the set of routes from m to n.

(i) Link flows, f_l, for each $l \in \mathcal{L}$: the average number of vehicles per hour, for each month/hour.

(ii) Origin/destination (O/D) counts, Q_{mn}, for each $(m, n) \in \mathcal{N}^2$: the average number of vehicles (per hour or day) travelling from m to n.

(iii) Route flows, h_{mnr}, for each $(m, n) \in \mathcal{N}^2$ and $r \in \mathcal{R}_{mn}$: the number of vehicles travelling from m to n taking route r.

(iv) Link speed functions, $v_l(f)$, for each $l \in \mathcal{L}$: the average vehicle speed as a function of the link flow. The average link travel time is of course $t_l(f) = \frac{\texttt{length}(l)}{v_l(f)}$.

It is important to bear in mind that information calculated from data using a model, such as the traffic assignment model, rather than directly measured also depends on assumptions implicit in the model, which may be overly strong.

6.3.1 Transition Probabilities

Since the model represents a high level of abstraction, the observable part of a state consists only of vehicle location, *i.e.* the network link it occupies. Because vehicle movements very much depend on the behaviour and intentions of the driver, a state typically also has an unobservable part. Which unobservable state variables the model contains depends on the information available for its construction. We present a number of different alternatives.

In the model descriptions, we use $\texttt{start}(l)$ and $\texttt{end}(l)$ to denote the starting and ending nodes of link l, $\texttt{length}(l)$ for the length of link l, and $\texttt{in}(n) = \{l \mid \texttt{end}(l) = n\}$ and $\texttt{out}(n) = \{l \mid \texttt{start}(l) = n\}$ for the sets of incoming and outgoing links to node n, respectively. Also, $\texttt{tt}(l)$ and $\texttt{tt}(m, n)$ denote the average travel time in link l and by the best route between nodes m and n, respectively.

Model #1

If route flow distributions are known, the unobservable state variable may be the route that the vehicle is following. The route determines the choice of next link at every node, *i.e.*

$$P((l,r),(l',r)) = \begin{cases} 1 & \text{if } l' \text{ follows } l \text{ in } r \\ 0 & \text{otherwise} \end{cases} \tag{6.14}$$

Given only one observation of the vehicles location, l, the most likely initial state distribution assigns to each route r such that $l \in r$ a probability equal to the fraction of the total flow on link l consisting of the flow along r:

$$\mathbf{P}(\mathtt{route} = r) = \frac{h_{mnr}}{f_l} \tag{6.15}$$

(since $f_l = \sum_{\{r \,|\, l \in r\}} h_{mnr}$).
The principle is easily generalized to a sequence of observations. Problems, however, arise if the sequence of observed locations is not consistent with any route that has non-zero flow. This is particularly likely to happen if route flow distributions are calculated using the traffic assignment model, which assigns flow only to optimal routes. A way to deal with this problem is to "fall back" on one of the models described below, since link flows can be easily calculated if route flow distributions are known.

Model #2

If link flows and volume/delay functions are known, the travel time in each link can be calculated. In this case, the unobservable state variable may be the vehicles destination (a node in the network), and the vehicle is assumed to favor links that shorten the remaining travel time to its destination.

Let $N(l)$ abbreviate $\mathrm{out}(\mathrm{end}(l))$, *i.e.* the set of possible successors links to a link l. For each $l' \in N(l)$,

$$P((l,n),(l',n)) = \frac{\left(\frac{1}{\mathtt{tt}(l')+\mathtt{tt}(\mathrm{end}(l'),n)}\right)^a}{\sum_{l'' \in N(l)} \left(\frac{1}{\mathtt{tt}(l'')+\mathtt{tt}(\mathrm{end}(l''),n)}\right)^a} \qquad a \geqslant 1 \tag{6.16}$$

i.e. the probability of the vehicle continuing along link l' is inversely proportional to the travel time to its destination if it were to take l' and continue optimally from the end of that link. For $l' \notin N(l)$, $P((l,n),(l',n)) = 0$. The "navigation constant, a, determines the strength of the tendency to navigate optimally (in the limit $a \to \infty$, only the optimal choice has non-zero probability).

In this model, a single observation of the vehicles location provides no information about the likelihood of different destinations. Given a series of observations, a simplified form of estimation algorithm 6.4 can be applied. If also origin/destination counts are known, the *a priori* likelihood of each destination may be made proportional to the volume of vehicles travelling to that node. Total link flows alone give no indication of the *a priori* probability of any destination, since a node that has a large inflow may have an equally large, or even larger, outflow.

Model #3

The two models described above both rest on fairly strong assumptions about the behaviour of the driver in choosing links to follow. If link flows are known, an alternative is to assume only that the vehicle "goes with the flow", *i.e.* to choose independently at each node a link l with probability equal to the fraction of flow into the node that leaves the node via l:

$$P(l, l') = \frac{f_{l'}}{\sum_{l'' \in N(l)} f_{l''}} \tag{6.17}$$

for $l' \in N(l)$ and zero elsewhere. This is a weak model: the choice of link leaving a node most likely depends on the link by which the vehicle arrived at the node, but this dependency can not be discerned from link flows alone. Since links typically represent "directed streets", a simple improvement to equation (6.17) is to exclude from $N(l)$ the link that is "opposite" to l, *i.e.* that represents travelling back along the same street.

In this model, there is no unobservable state and consequently no need for estimation. The disadvantage is that no additional information can be gained from having more than one observation: the initial state distribution is always concentrated to the last observed vehicle location.

6.3.2 Event Time Distribution

At the level of abstraction that we have adopted, the time between state transitions of the embedded Markov chain corresponds to the travel time of the link currently occupied by the vehicle. Knowledge about link travel time is typically available in the form of a volume/delay function, which gives average speed as a function of the volume of traffic in the link. Since, unfortunately, it tells us nothing about the shape and magnitude of deviations from the average, we have no alternative but to assume a distribution.

Let V_l be the random variable representing vehicle speed in link l. Assuming that V_l is normally distributed around the average $v_l(f)$ (where f is the current link flow) with a standard deviation of σ_l, the link travel time is a random variable $T_l = \frac{\mathtt{length}(l)}{V_l}$, with the distribution

$$\begin{aligned} F_l(x) = \mathbf{P}(T_l \leqslant x) = & \\ = & \ \mathbf{P}(V_l \geqslant \frac{\mathtt{length}(l)}{x}) \\ \approx & \ 1 - \mathbf{P}(V_l \leqslant \frac{\mathtt{length}(l)}{x}) \\ = & \ 1 - \Phi(\frac{\frac{\mathtt{length}(l)}{x} - v_l(f)}{\sigma_l}), \end{aligned} \tag{6.18}$$

$x > 0$, where $\Phi(\cdot)$ denotes the standard normal distribution function[2]. Distribution (6.18) may be somewhat complicated, but to apply the phase tranformation we need only to be able to evaluate it at certain points. The phase count probabilities, $p_\delta(k)$, according to equation 6.7, then become

$$p_\delta(k) = \Phi(\frac{\frac{\mathtt{length}(l)}{(k-1)\delta} - v_l(f)}{\sigma_l}) - \Phi(\frac{\frac{\mathtt{length}(l)}{k\delta} - v_l(f)}{\sigma_l}) \tag{6.19}$$

for $k > 1$ (for $k = 1$, the first term is approximately 1).

Quite possibly, the particular vehicle of interest does not behave as the average, but with only one observation there is little else we can assume. With a series of observations, however, a better model of the vehicles speed may be estimated.

The principle is the same as for any statistical model, described in section 2.7: we assume that the vehicle speed varies according to some parameterized probability distribution, and estimate the most likely parameter values, given the observations. There are many possible models to choose from, but the linear regression model, described in section 2.7.1, is simple and probably sufficient. In this model, V_l is normally distributed around a mean $\alpha v_l(f) + \beta$, where $v_l(f)$, the "default" average speed of the link, is the dependent variable. The standard deviation may be fixed as above, or estimated (if the number of observations is small, a fixed value is likely to work better, since an estimate will be very uncertain).

6.4 Evaluation

As in chapter 5, we discuss each of the three points representation, computation, and integration. A comparison of the two model designs is the topic of chapter 7.

[2]The derivation of formula (6.18) is valid only if $V_l > 0$, which is reasonable since it represents the vehicles speed, but the normal distribution assigns positive probability to negative values. We may either assume that $v_l(f)$ and σ_l are such that this probability is negligable, or use a different distribution for V_l.

6.4.1 Representation

The assumptions of the Markov jump process may seem restrictive, but, as shown above, can for the most part be circumvented: almost any reasonable Markov process can be transformed into a Markov jump process, using the phase method, albeit with some loss of precision in the event time distribution, and even a non-Markovian stochastic process can be handled, by encoding sufficient process history in the state.

The various model constructions described above represent our best effort at making use of any statistical knowledge available. The probabilities expressed by the models are, however, not as firmly grounded in statistical truth as they may seem, since each model construction incorporates certain assumptions, whether directly, as *e.g.* in model types #2 and #3, or indirectly, as in model type #1 where the assumptions are introduced in the calculation of route flow distributions rather in their use in the model construction.

6.4.2 Computation

The algorithm for computing future distributions is exponential in the dimension of the state space, but polynomial in the prediction horizon and almost independent of the amount of uncertainty in the initial distribution or the transition probabilities. Due to the encoding of event times as phases, however, the size of the state space increases with event time "span", *i.e.* the range of values of k for which $p_{F_x,\delta}(k) > 0$. The complexity of inference depends mostly on the number of observations, which is in our application generally small.

Note also that for the reidentification problem described in scenario 1, it is not necessary to compute a probability distribution over all model states: only the probabilities of states in which the vehicle is in a particular location are of interest, and this enables more efficient algorithms. For instance, we may enumerate only the set of routes from the last observed location to the present, and compute for each route its probability, and the probability of the vehicle travelling that route in the observed time.

6.4.3 Integration

While the requirements of the reidentification scenario are easily met by the general prediction algorithm, and maybe more efficient specializations, the search planning problem of scenario 2 is not as easily dealt with: algorithm 6.3 computes the probability distribution at a *point* in time, but to effectively plan a search strategy we need to know the probability of the vehicle appearing in a certain location during an interval of time, something far more complicated to calculate.

Chapter 7

Experimental Evaluation

The three points we suggested in chapter 3 to consider when evaluating a model design were representation, computation and integration with the application. Each of them has been discussed in connection with the two designs, in sections 5.5 and 6.4.

This chapter complements the picture by reporting on experiments made with both model designs. The next section describes an experimental environment in which prototype implementations of the two model designs have been tested, while section 7.2 describes test scenarios and results. Finally, some experiences gained, in the construction of the experimental environment as well as from the test scenarios, are described in section 7.3.

7.1 Experimental Environment

The experimental setup aims to test our two model designs through as much of the models life as possible: through model construction and maintenance to use in the two scenarios described in chapter 4. The prediction systems, however, have not been integrated with the UAV control architecture, and thus were not actually used in planning or reidentification.

7.1.1 Road Network and Traffic Data

The road network used in experiments presents a simplified view of central Norrköping, a medium-sized swedish city. Available traffic data consisted of measurements of yearly day average flow, made during 1999 and 2000, in 140 roads[1]. 82 measurements were made in two-lane roads and give only the total flow in both directions,

[1]Thanks to Linda Wahlman at Norrköpings Komun, who provided data and answered a lot of questions.

i.e. the sum of flow in two directed links. Network topology, link lengths and road type classifications were obtained from an ordinary map.

The network contains 139 nodes, 28 of which are terminal, and 414 links. The links included in the network were chosen to be those for which flow measurements were available, plus what was needed to achieve a reasonable level of connectivity. As a consequence, the network contains mostly main entry- and thoroughfares. Clusters of smaller streets, *e.g.* in residental areas, were abstracted into one or two terminal nodes. Because the network consists mostly of urban roads, only 14 different road types, according to the classification by Matstoms, Jönsson & Carlsson (1996), were used.

Network Flow Assignment

Even though a large number of nodes are non-terminal, *i.e.* nodes whose inflow/outflow sums to zero, taking only flow conservation constraints into account leaves the network flow highly underconstrained. To select among the many consistent flow assignments, we chose to minimize the origin and destination counts of each terminal node, under conservation constraints and a weak acyclicity constraint. The rationale for this choice is basically that there is no reason to believe that there is significantly more traffic in unmeasured links than in those measured. The acyclicity constraint states that the flow in a link l leaving node n is no greater than the sum of flow into n along all links except the one opposite to l, if such a link exists. To avoid the possibility of inconsistency, this constraint was applied only to terminal nodes. Finally, each measurement of the sum of flow in two opposite links gives rise to an obvious constraint.

The resulting optimization problem is linear, with variables O_n and D_n for the origin and destination count of each terminal node, and f_l for the flow in each link l:

$$\min \sum_{n \in \mathcal{N}_T} O_n + D_n$$

$$\begin{aligned}
\sum_{l=(n',n)\in\mathcal{L}} f_l - \sum_{l=(n,n')\in\mathcal{L}} f_l &= D_n - O_n \quad \forall n \in \mathcal{N}_T \\
\sum_{l=(n',n)\in\mathcal{L}} f_l - \sum_{l=(n,n')\in\mathcal{L}} f_l &= 0 \quad \forall n \in \mathcal{N}_Z \\
f_l &\leqslant \sum_{l'=(n',n)\in\mathcal{L}, l'\neq\bar{l}} f_{l'} \quad \forall n \in \mathcal{N}_T, \forall l = (n,n') \in \mathcal{L} \\
f_l + f_{\bar{l}} &= \texttt{flow}_{l,\bar{l}} \quad \forall l : l + \bar{l}\,\text{measured}
\end{aligned}$$

where $\mathcal{N}_T$ and $\mathcal{N}_Z$ are the sets of terminal and non-terminal nodes, respectively, and $\bar{l}$ denotes the link opposite to l. Variables representing links for which directed measurements were available were of course replaced by the measured value.

The approach seems to yield reasonable network flow assignments. The calculated node origin and destination counts, however, are very small, most likely far smaller than actual values (implying that under the calculated assignment, a lot of traffic "circles"). Under the resulting network flow, the average speed calculated according to V/D functions is for most links close to the free flow speed. This is not so surprising, since congestion effects typically appear only at peak traffic, with volumes around 2 – 4 times the yearly day average.

7.1.2 Prediction Systems and Models

The basic prediction and query evaluation algorithms for expectation/normality models are implemented in a system called PRECOG. To facilitate model development and experimentation, the system is not application specific, but uses instead a general language for model specification. A short manual for the system is in appendix A.

The prediction system based on Markov models is called MMP. Because of the arbitrary nature of the functions defining transition probabilities and event time distributions, creating a general language for model specification was deemed too complicated, and the different models used are instead hardcoded into the system. Currently, network data is also hardcoded into the system, but it may quite easily be changed to read data from file, or from the geographic information system that is part of the UAV on-board system.

Both prediction systems work "off-line", *i.e.* take as input a set of initial facts and observations (and, in the case of PRECOG, the model) and a set of queries/prediction requests, and produces answers in a variety of forms.

Expectation/Normality Models

Three different models, **A**, **B** and **C**, were used in the experiments. The transition system part is the same in all models, and comprises essentially the transition rules and state constraints found in examples 5.2 and 5.3.

Model **A** contains the following expectations:

(α) The vehicle does not turn to the opposite direction on the road it is travelling in:

$$\begin{array}{l}\Box_{[0,\infty]}\texttt{let}\, r_0 = \texttt{loc in} \\ \quad \bigcirc_{[0,\infty]}(\texttt{loc} = r_0 \vee \texttt{loc} \neq \texttt{opposite}(r_0))\end{array}$$

(β) The vehicle keeps within $-5/+15$ km/h of the average speed for the road it is in:

$$\begin{aligned}
&\Box_{[0,\infty]}\mathtt{let}\, r_0 = \mathtt{loc\ in} \\
&\quad (\bigcirc_{[0,\infty]}(\mathtt{loc} = r_0 \vee \mathtt{let}\, r_1 = \mathtt{loc\ in} \\
&\qquad \Box_{[0,\mathtt{length(loc)/(avg\_speed(loc)+15)}]}\mathtt{loc} = r_1) \wedge \\
&\quad \Diamond_{[0,\mathtt{length(loc)/(avg\_speed(loc)-5)}]}(\mathtt{loc} \neq r_0 \vee \neg\mathtt{moving}))
\end{aligned}$$

Since the vehicle speed is not a state variable, the expectation formula places instead lower and upper bounds on the time that it remains in the road.

(γ) The vehicle only stops at its destination:

$$\Box_{[0,\infty]}(\mathtt{end(loc)} = \mathtt{dst} \vee \mathtt{moving})$$

(δ) The vehicle chooses the shortest route to its destination, within a constant error:

$$\begin{aligned}
&\Box_{[0,\infty]} \\
&\quad (\mathtt{length(loc)/avg\_speed(loc)}) + \mathtt{distance(end(loc),dst)} \leqslant \\
&\quad \min_{\substack{r_1:\mathtt{start}(r_1)\\ =\mathtt{start(loc)}}}((\mathtt{length}(r_1)/\mathtt{avg\_speed}(r_1)) + \mathtt{distance(end}(r_1),\mathtt{dst})) + 2
\end{aligned}$$

Model **B** is identical, except that expectation (β) is strengthened to constrain the vehicle speed to $-2.5/+2.5$ km/h of the average speed for the road.

In model **C**, the state variable `dst`, representing the vehicle destination, is removed, along with expectations (γ) and (δ). The condition that the vehicle stays moving is included in (α), and the stronger version of expectation β is used.

The complete models may be found in appendix B.

Markov Models

Model types #2 and #3 described in chapter 6 have been implemented in the MMP system. The type #1 model was not implemented, since origin/destination counts were not available, and hence route flow distributions could not be calculated. In the type #2 model, all destinations (terminal nodes) are given equal *a priori* probability, since the destination counts calculated by the network flow assignment appear quite unreasonable. For the navigation constant in equation (6.16), $a = 1$ and $a = 3$ was used.

The event time distributions are calculated according to equation (6.18), using the V/D functions by Matstoms *et al.* (1996) with the road classification and the calculated link flows to obtain the average speed. Since no information about speed variations was available, the coefficient of variation is fixed: the standard deviation then equals the mean times the coefficient. Two different coefficient values, 0.1 and 0.25, were, rather arbitrarily, chosen. Parameter estimation for the event time distribution was not used.

7.2 Experiment Scenarios and Results

Two different scenarios were used: in the first, predictions were made with only an initial location, using seven different models (expectation/normality models **A**, **B** and **C**, and Markov model types #2 and #3, each with coefficient of variation 0.1 and 0.25), while in the second, predictions were based on two observations. In the first scenario, a navigation constant $a = 1$ was used for model #2, while in the second two values, $a = 1$ and $a = 3$, were used.

Because we lack the means to measure the accuracy of predictions against reality, two other measures are considered. The first is "spread", *i.e.* how many different vehicle locations are considered to be likely at the prediction horizon. Since there is no uncertainty about the vehicles initial location, this gives an indication of how the uncertainty grows with the prediction horizon. The second measure is simply the computation time.

For the Markov models, it is somewhat difficult to determine how many locations are "likely", since they yield a probability distribution over all locations. For these models, spread is measured by a number n such that the n most likely locations, starting from the most likely and continuing down, (a) add up to a total probability of 0.9, and (b) the total probability of the $n + 1$ most likely location is less than 1% more than that of the n most likely, respectively.

7.2.1 Scenarios

For the first scenario, 10 different initial locations were chosen (more or less at random), and predictions made with a horizon of 2 – 3 minutes, using each of the seven models. In the second, one initial location and 5 different locations for a second observation 3.5 – 4.5 minutes later were chosen. Predictions were made with a horizon of 6 – 7 minutes, using models **B**, **C**, #2 and #3 (with coefficient of variation 0.1 for both). The observed locations were chosen to be consistent with models **B** and #2.

7.2.2 Results

Results of the two experiment scenarios are summarized in tables 7.1 and 7.2. For Markov models, spread measures (a) and (b) are both shown, although it turned out they are roughly the same.

That spread increases with the prediction horizon is no surprise, though it appears to grow more quickly in expectation/normality models. The large variation seen in the results from all models is also to be expected, since the number of possible routes varies across the road network.

In the first scenario, there is not much difference in spread between models incorporating the vehicles destination and models that do not, *e.g.* between models **B** and **C**, or between models #2 and #3. This is to be expected, since with only one

Model ($c.o.v.$)	Spread at $H = 2$ min.	max.	avg.	Spread at $H = 3$ min.	max.	avg.	Time (sec.) avg.	med.
A	3	56	12.2	3	133	28.2	803.2	53.0
B	3	26	7.0	3	73	16.0	163.0	39.0
C	3	26	7.2	3	73	16.3	8.5	6.0
		(a/b)			(a/b)			
#2 (0.1)	2/4	26/20	8.6/8.7	4/4	43/30	13.6/13.7	6.0	
#2 (0.25)	2/4	24/19	8.1/8.4	2/4	40/29	13.4/13.5	9.1	
#3 (0.1)	2/3	18/14	5.7/7.1	2/4	34/24	9.7/10.5	5.3	
#3 (0.25)	2/3	16/15	5.6/6.8	2/4	32/20	10.2/11.4	7.9	

Table 7.1: Results from the First Scenario

Model	Spread, $H = 6$ min.	max.	avg.	Spread, $H = 7$ min.	max.	avg.	Time (sec.) avg.	med.
B	1	9	4.6	1	11	5.2	15.4	14.0
C	1	11	5.1	1	15	7.8	6.0	4.0
		(a/b)			(a/b)			
#2 ($a = 1$)	1/4	9/14	5.2/8.6	4/7	14/15	9.0/12.0	9.8	
#2 ($a = 3$)	1/4	9/14	5.0/7.8	4/7	13/14	8.4/11.2	8.8	
#3	1/3	8/11	5.8/7.2	3/5	13/15	8.0/10.2	9.0	

Table 7.2: Results from the Second Scenario. The two versions of model #2 differ in the navigation constant, a.

observation of the vehicles location very little about its destination can be infered. In the second scenario, model **B** shows less spread than model **C**, and model #2 with a navigation constant $a = 3$ less than with $a = 1$, as expected, but, somewhat surprisingly, model #3 has even less spread than both type #2 models.

Another surprise is that increasing the coefficient of variation does not cause a more noticable increase in spread (and in some cases even causes a decrease), since it should increase uncertainty about how long the vehicle remains in each link. By contrast, model **A** shows a much greater spread than model **B**: the only difference between them is that the stronger expectation β in model **B** reduces uncertainty about time.

Concerning computation time, prediction with the Markov model design is in almost all cases faster, in most cases much faster, than with expectation/normality models[2]. The computation time for expectation/normality prediction is also subject to a much greater variation, as the difference between mean and median shows, while for the Markov model design deviations from the average are small (a second at most).

7.3 Experiences

The main point of the experiment was not to produce the numbers in tables 7.1 and 7.2, which are largely artifacts of the actual models and the data they were built from, but rather to test the two model designs throughout the process of model acquisition, maintenance and use.

Because the prediction systems have not been integrated with the UAV control architecture, the actual use of predictions in planning and decision-making could not be tested. This is a weakness, but unavoidable since presently, the UAV can not be operated outside designated testing areas, and hence can not observe real traffic.

7.3.1 Model Acquisition and Maintenance

Writing expectation/normality models **A**, **B** and **C** was rather straightforward, although formulating expectations in MITL requires some familiarity with the language. An important point, that the experimental results do not show, is that changing the form of an expectation formula can result in quite different prediction computation time.

The mundane work of converting a map to a node/link network representation aside, the most difficult and time consuming part of model construction was to invent, and apply, the solution to network flow assignment. Acquisition in the Markov model design is "data intensive", and it is interesting to note that the most work was spent

[2]Note, however, that with model **C**, the computation time is roughly the same as with both Markov models. This is most likely because model **C** has no initially undetermined state variables and PRECOG therefore constructs only a single prediction tree, while models **A** and **B** have on average 18 consistent initial states and a tree is built for each. A more compact representation of partially known states might improve this, as discussed in section 5.5.2.

essentially compensating for the lack of data, *i.e.* flow measurements.

It is questionable whether the experiment as carried out involved any model maintenance to speak of. Since predictions could never be confronted with reality, there was no way to discover discrepancies and therefore no reason to correct the models. Nevertheless, an illustrating example is provided by the lack of difference caused by changing the variance of the event time distribution in the Markov models. A significant change to event timing in the Markov models seems to require replacing the event time distribution (equation 6.18) by a different distribution altogether. By contrast, the change in timing uncertainty between expectation/normality models **A** and **B** is extremely simple (in fact, model **B** was added to the test after it was discovered that Markov model predictions showed very small timing uncertainty). Changing the navigation constant in the type #2 model, on the other hand, does affect predictions, in the expected direction, although not by much.

7.3.2 Computational Efficiency and Predictability

The Markov model design yields, on average if not in every case, faster predictions than the expectation/normality model design. But, the computation time for prediction with Markov models is also quite predictable, which may be an even more important property if prediction is going to be part of an on-line, maybe even real time, decision making process, *e.g.* reidentification. By changing the order of summation in equation (6.4), the procedure for computing the probability distribution of a Markov jump process at time $t + \Delta$, given the distribution at t, becomes

$$\pi_{t+\Delta}(x) = \sum_{k \geqslant 0} F_P(k, \lambda\Delta) \left(\sum_{x' \in \mathcal{X}} \pi_t(x') P^k(x', x) \right)$$

where $F_P(k, \mu)$ denotes the Poisson probability density function with mean μ. The inner sum needs only be evaluated for values of k such that $F_P(k, \lambda\Delta)$ is negligably greater than zero. Furthermore, for $\lambda\Delta$ sufficiently large ($\lambda\Delta \gtrapprox 15$), the Poisson distribution is approximately normal with mean and variance equal to $\lambda\Delta$, which means that the range of interesting k values can easily be upper and lower bounded. By evaluating the outer sum in order of increasing k and storing intermediate values of P^k, the calculation in the inner sum can be simplified to

$$P^k(x', x) = \sum_{y \in \mathcal{X}} P^{k-1}(x', y) P(y, x)$$

which is computable in constant time (since the state space may be considered fixed). Thus, the time to compute a prediction to a given horizon can itself be predicted, with very small overhead.

The time to compute predictions with expectation/normality models, on the other hand, is highly unpredictable: in the results of the first experiment scenario, there is a weak correlation between the number of possible consistent initial states and the computation time, but even this fails in the presence of additional observations.

Chapter 8

Conclusions

The aim of this inquiry, as stated in the introduction, has been to investigate possible designs for predictive models and to assess the suitability of different designs in particular cases.

In part I, this was done in a very general setting. From a survey, certainly not exhaustive, of control and reasoning tasks involving prediction, several choices that may have to be made in the design of a predictive model were identified, along with alternatives for each. The setting in part II, in contrast, was highly specific: here, we considered the problem of predicting the movements of a vehicle in a road network, over a relatively large time scale and given scant information about the vehicle in question.

8.1 Vehicle Movement Prediction

At the end of chapter 4, several possible model designs were sketched for the vehicle movement prediction problem. Two designs were then developed in detail and compared. The two differ mainly in the way uncertainty is represented: in the expectation/normality model design, it is a qualitative measure based on "common sense", or normal case, knowledge, while in the Markov model design uncertainty is represented by probabilities, derived from traffic flow statistics.

The results of the comparison are not conclusive. From a computational point of view, the Markov model design has obvious advantages, being both highly predictable and the faster most of the time, but there are also difficulties: data requirements for the ideal model construction (based on route flow distributions) are unrealistic, and atempts to compensate for lack of data by introducing assumptions resulted in "opaque" models. Concerning the expectation/normality model design, the main flaw apparent in experiments is that the time to computate predictions is highly varied, and most of the time unrealistically long. However, due the short model lifetime, our experiments can not really be said to have tested representational properties such as ease of maintenance.

Development of the two model designs has been largely a matter of applying existing theory and techniques: the Markov jump process, timed transition systems and MITL are all well known, although they have, as far as we are aware, not been applied to a "pure" prediction problem like this before. The extension of the MITL progression algorithm to dealing with only constrained event times, however, is new.

8.2 A Theory of Predictive Model Design?

Can the discussion in part I be said to constitute a general theory of predictive model design? Perhaps, but if so, it is a very weak theory. Although we identified, from the examples surveyed, alternative representations of time and uncertainty, methods of model acquisition and of computing predictions, the choice of each is almost always constrained by the application. The knowledge available to build a model from and the form of predictions required limit, and often completely determine, the applicable methods of model acquisition and modes of computation, and these in turn are strongly tied to the representational choices. The design choices and alternatives listed in chapter 3 are perhaps best viewed as a catalouge, not exhaustive, of designs and methods that have been used, to serve as a source of inspiration for the design of new predictive models.

In cases where there are several plausible design candidates, the question of which is the better is even more difficult to answer. For evaluation and comparison, we suggested three categories of properties of a model design to consider: representational, computational, and matching the conditions and needs of the application. As shown by the case study of designing a model for the vehicle movement prediction problem, simple answers should not be expected for any of them. The results of a single case study, no matter how thoroughly carried out, are no basis for general conclusions, but perhaps comparison of alternative model designs across a wide range of applications may unearth some more generally valid principle.

8.3 Future Work

With an aim as ambitious as that of this inquiry has been, we must inevitably leave much room for improvement, both in the case we have studied and in a more general context.

The Markov and expectation/normality model designs have some traits in common: they are both event based and represent states at a high level of abstraction, and, perhaps most importantly, the process of computing predictions is in both cases a, more or less, exhaustive enumeration of developments. As mentioned in the discussion of possible model designs at the end of chapter 4, another possibility would be sampling, at random or according to some other principle, the set of developments, which would allow for a more detailed state representation and more granular view of time. Thus, for even the study of our chosen case to be complete, at least one more model design should be developed, evaluated and compared with the other two.

We need also to examine more cases, to look at what kinds of models are being used for prediction and if there are any viable alternatives. Several examples of problems involving prediction are found in the context of the WITAS UAV project: the reidentification problem occurs also on a much smaller time scale, where a whole different set of model designs are likely to be appropriate, and there are also problems of tracking control in which prediction is used, or could be used, both in the control of the helicopter and camera and at the discrete event level of the control system.

For the search planning problem, prediction is only one part of the solution: the other part, planning a search strategy based on the information provided by prediction, is most likely beyond the capability of AI planning of today, and a research problem in itself.

Appendix A

The PRECOG System - A Short Manual

The PRECOG system is a prototype implementation of the basic prediction algorithm for expectation/normality models. The system works in "off-line mode": it reads a model description, including initial conditions and any observations, generates a prediction forest to a specified prediction horizon and level of normality and finally presents the results in a variety of forms. The system also implements the basic query evaluation algorithm described in section 5.4.

A.1 State Language and Model Description Syntax

For state formulas (occuring in transition rule and state constraint preconditions, expectations, observations, queries and more), PRECOG uses a finite, functional language which offers a reasonable trade-off between expressivity and efficiency. Quantifiers, and parameterized state variables, transition rules and state constraints are allowed, but are treated only as a shorthand and converted to ground instances internally.

A model description consists of a set of declarations, each ended by a period ("`.`"). Declarations are evaluated in the order they are read. Comments in model description files are indicated by `#` and last to the next end-of-line.

A.1.1 Domain Sorts and State Variables

The model may use values of three sorts: Boolean values, written `T` and `F`, respectively, numeric values (floating point) and a finite set of objects. There is actually a fourth sort, "any", containing only one value, written "_", which has the property that it is not equal to itself.

Elements of the object sort have to be enumerated, which is done by the declaration

```
object obj_1, obj_2, ..., obj_n.
```

The enumeration can be split into several `object` declarations.

The state is represented by a collection of state variables, which may take any domain value. Variables are not sorted, and so may change the sort of value they take during a development. A state variable is declared by

```
state <name> {<param_1> ... <param_n> : <condition>}.
```

where `<param_1>, ..., <param_n>` are local variables (any symbol not declared to be an element of the object sort or a state variable is assumed to be a local variable) and `<condition>` a formula in these local variables. The declaration creates distinct state variables for each assignment of domain objects to the parameters that makes the condition true.

For example,

```
state location.
state light_state {l : is_link(l) & has_traffic_light(l)}.
```

creates one state variable `location`, and one state variable `light_state(l)` for every object `l` in the domain for which `is_link(l) & has_traffic_light(l)` evaluates to true.

A.1.2 Static Functions

The static part of the model description is given by a collection of user-defined static functions. Functions, like state variables, are not sorted, so that a function may have values of different sorts for different arguments. The same function may also be used with a varying number of arguments. Function arguments, however, are restricted to be of the object sort.

Function declarations are writen as an equality between the function expression and the value. For example,

```
link(l12) = T.
ends_at(l12) = n2.
length(l12) = 24.6.
```

sets the values of the functions `link`, `ends_at` and `length` for the argument `l12` to true, the object value n2 and the numeric value 24.6, respectively.

The default value of all functions is "_", so this value will be returned if a function is evaluted with a set of arguments for which it has not been given a value.

Defined Functions

It is also possible to define a static function as an "abbreviation" for an expression. This is written

```
define <function>(<param_1>, ..., <param_n> : <condition>) =
  <expression>.
```

where `<param_1>, ..., <param_n>` are local variables. A defined function is defined only for arguments that satisfy the `<condition>` formula.

A.1.3 Expressions and State Formulas

Expressions are built from constants, state variables, local variables, static functions and a set of built-in functions. The only primitive relation in the language is equality between expressions, though boolean valued expressions may be also used as atomic formulas (shorthand for "`<exp> = T`"). Complex formulas are built using the boolean connectives, negation, conjunction and disjunction (implication can be used as an abbreviation).

Static function expressions are written `<function>(<arg_1>, ..., <arg_n>)`, same as in their declaration, and the rest of the expression syntax is for the most part straightforward. The built-in functions have some syntactical peculiarities, however, detailed in section A.1.4. Note that any symbol not declared to be an element of the object sort or a state variable is automatically made a local variable.

The logical connectives are written "`!`" (negation), "`&`" (conjunction), "`|`" (disjunction) and "`->`" (implication). Universal and existential quantifiers are written

```
A {<var_1> ... <var_n> : <condition>} <formula>
E {<var_1> ... <var_n> : <condition>} <formula>
```

where the local variables `<var_1> ... <var_n>` and the formula `<condition>` are treated as in state variable declarations. Note that they will be expanded into a conjunction resp. disjunction, of instantiated copies of the `<formula>`.

A.1.4 Built-In Functions

PRECOG has the following built-in functions:

Arithmetic Operations and Relations Apply to numeric valued arguments are result in numeric or boolean values. They are written `+`, `-`, `*`, `/`, `<`, `>`, `<=` and `>=`. Note that unary `-` has higher precedence than binary, so `x-2` is not a valid expression (it is read as `x` followed by the integer constant `-2`). It must be written `x - 2` (with a space separating the `-` from `2`).

Min and Max The functions `min` and `max` can be used in two forms: The first form is written `min <exp_1>, <exp_2>` (resp. `max <exp_1>, <exp_2>`), where `<exp_1>` and `<exp_2>` are numeric valued expressions, and returns the minimum (resp. maximum) of the two expressions.

The second form is written `min {<var> : <condition>} <exp>` (and analogously for `max`) and returns the minimum value of `<exp>` for any instantiation of the local variable `<var>` that satisfies the state formula `<condition>`. They default to $-\infty$ and ∞, respectively, if no instantiation satisfies `<condition>`. In this second form can also be used the operators `argmin` and `argmax`, which return the value of `<var>` that achieves the minimum and maximum values instead.

Select The `select` operator returns an element of the object sort that satisfies a specified condition, provided a *unique* such element exists. The syntax is `select {<var> : <condition>}`, where `<var>` is a local variable and `<condition>` a state formula. If no unique element satisfying the formula exists, the expression returns "any" (`_`).

Conditional The expression `if <formula> <exp_1>; <exp_2>` returns the value of `<exp_1>` if `<formula>` is true, and the value of `<exp_2>` if not.

Type Tests The expressions `is_any <exp>`, `is_bool <exp>`, `is_obj <exp>` and `is_real <exp>` returns true iff the value of `<exp>` is of the respective sort, and false otherwise.

A.1.5 Transition Rules and State Constraints

The precondition of a transition rule or state constraint is a state formula and the time bounds are arbitrary numeric valued expressions. The postcondition of a transition is restricted to be a conjunction of equalities between state variables and expressions, which are interpreted as assignments.

All transition rules and state constraints are labeled with a name, and for descriptive economy may have parameters (written the same way as for state variables, above). The syntax is

```
<name> {<param>* : <param condition>}:
 <precondition> => [<min>,<max>] <postcondition>.
```

for transition rules and

```
<name> {<param>* : <param condition>}:
 <state condition> : <max>.
```

for state constraints (the braces following the name may be left out if the transition rule or constraint has no parameters). For example, the declaration

```
drive_road {l : is_link(l)}:
 connected(loc,l) & at_start & moving
 => [(length(loc)/120), INF] loc(c) = l.
```

define one transition rule `drive_road(l)` for every domain object that represents a link.

A.1.6 MITL Formulas and Expectations

MITL formulas are built from state formulas, modal operators, standard connectives and the special `let` operator. The modal operators are written `A[<min>,<max>] <formula>` (*always*), `E[<min>,<max>] <formula>` (*eventually*), `X[<min>,<max>] <formula>` (*next*) and `(<formula> U[<min>,<max>] <formula>)` (*until*), where `<min>` and `<max>` are arbitrary numeric valued expressions. Note that the until operator requires a pair of parenthesis. The variable binding operator is written

```
let <var> = <exp> in <formula>
```

Expectations are declared as follows:

```
expect <formula>.
```

Only a linear hierarchy is supported, and the order of declaration determines position in the expectation hierarchy (the first declared being the lowest).

A.1.7 Initial Condition and Observations

There are two forms for declaring facts about the initial situation:

```
init assign <var_1> = <val_1>, ..., <var_n> = <val_n>.
init select <formula>.
```

The first assigns specific values to some subset of state variables, while the second declares a condition that should hold.

Because PRECOG works in "batch mode", observations are also part of the model description. They are declared with

```
obs [<time>] <formula>.
```

A root node in the prediction forest will be created for every assignment of elements of the object sort to the state variables that have not been initialised by an `init assign` declaraion and that satisfies the initial condition. In the construction of the prediction forest, for every observation, any node in which the observation `<formula>` is false is constrained to either end before or begin after the observation `<time>` (which may cause some nodes to become temporally inconsistent, and thus excluded from the forest).

An initial condition should not be declared as an observation at time 0.

A.2 Queries

Queries are also input to PRECOG as part of the model description. The query forms supported are the query formulas described in section 5.4, including normality as a modality, and a simpler form of value set query.

Query declarations end with a question mark (?) instead of a period. The syntax of query formulas is

```
N[<level>] A[<t_0>,<t_1>] <formula>?
N[<level>] E[<t_0>,<t_1>] <formula>?
P[<level>] A[<t_0>,<t_1>] <formula>?
P[<level>] E[<t_0>,<t_1>] <formula>?
```

where `<formula>` is a state formulas, `<level>` a zero or positive integer less than or equal to the greatest level of normality (*i.e.* the number of expectations) and `<t_0>` and `<t_1>` are constant numeric values. The normality specification can be abbreviated to `[]`, which is interpreted as "most normal".

Value set queries are declared

```
W[<level>] [<t>] <expression>?
```

which is interpreted as a request for the values of `<expression>` occuring at time `<t>` in at least one development normal at `<level>`.

A.3 Invoking PRECOG

As mentioned, PRECOG runs in "off-line" mode: it reads the model description, computes a prediction forest to a specified prediction horizon and level of normality, and presents the results. The program is invoked with

```
precog <command>* <file>* [-horizon <horizon>] [-normal <level>]
       <option>*
```

where possible commands are

`-t` Prints the part of the development forest normal at `<level>`.

`-p <horizon>` Prints the set of all developments (paths) up to `<horizon>` that are normal at `<level>`. Developments are expanded up to the point where the earliest possible ending time of the last state in the sequence is necessarily greater or equal to `<horizon>`.

`-b <min> <max>` Prints the set of all states in developments normal at `<level>` whose duration possibly intersects the interval `[<min>,<max>]`.

- `-q` Evaluates queries and prints the results (see section A.2 on queries above).
- `-r` Prints the root nodes.
- `-a` Prints the set of all nodes in the forest.
- `-m` Prints the model (after instantiation) and exits.

The most important options are

`-cache 0|1|2` Controls memory usage. The default cache level is 2, which means that the prediction forest is built up to the greatest horizon and lowest normality level needed before printing and query evaluation. At level 1, the forest is constructed "on the fly", as developments are enumerated. At level 0, the forest is also deleted on the fly, and reconstructed when needed. This slows down evaluation of multiple queries, but may be necessary if memory is tight.

`-count <n>` A trace printing option: prints a message for every `<n>`:th node created or deleted.

`-detail <n>` Controls the level of detail in printing. A value of 0 eliminates almost all output, while a value of 4 or more turns on the maximal volume of unnessecary information. The default is 1.

`-stat` Prints some statistics at the end.

The `-horizon`, `-normal` and `-cache` options may be abbreviated to `-h`, `-n` and `-c`, respectively.

There are more trace printing and formating options: run `precog -?` for a summary.

Appendix B

Expectation/Normality Vehicle Movement Models

B.1 The Model

Below is the expectation/normality model **A** created as part of the experiment. This is not the complete model: the road network description is in a separate file (comments in the model describe the required contents of the network description), as is a “scenario” containing initial condition, observations and queries.

```
# The following function are provided by the road network
# description. Note: unit of time is minutes!

link() = _.       # link(<r>) = T for all links
node() = _.       # node(<n>) = T for all nodes
starts_at() = _. # starts_at(<r>) = <n> starting node of link
ends_at() = _.   # ends_at(<r>) = <n> ending node of link
length() = _.    # length(<r>) = <num> length of link (km)
avg_speed() = _. # avg_speed(<r>) = <num> avg. speed in link (km/h)
terminal() = _.  # terminal(<n>) = T iff node <n> is terminal
distance() = _.  # distance(<n0>,<n1>) is the average travel time
                 # from <n0> to <n1> along the quickest route

# The model is written to allow for scenarios involving more than
# one car. The scenario has to set car(c) = T for each object c
# that represents a car.

car() = _.
```

```
# Type test macros. Note that defined functions are expressions,
# not formulas, and therefore can not use normal negation.

define is_car(c : T) = if (!is_obj c) F; if is_any car(c) F; T.
define is_link(x0 : T) = if (!is_obj x0) F; if is_any link(x0) F; T.
define is_node(x0 : T) = if (!is_obj x0) F; if is_any node(x0) F; T.

# The functions connected and opposite are defined in terms of
# starts_at and ends_at.

define connected(r1, r2 : T) =
 if !is_link(r1) F;
 if !is_link(r2) F;
 ends_at(r1) = starts_at(r2).

define opposite(r1 : is_link(r1)) =
 select {r2: connected(r1,r2) & connected(r2,r1)} r2.

# State variables:

state loc {c : is_car(c)}.
state at_start {c : is_car(c)}.
state moving {c : is_car(c)}.
state dest {c : is_car(c)}.

# Transition rules:

drive_road {c, r : is_car(c) & is_link(r)}:
 connected(loc(c),r) & at_start(c) & moving(c)
  => [(length(loc(c))/120)*60, INF]
 loc(c) = r.

drive_part_of_road {c, r : is_car(c) & is_link(r)}:
 connected(loc(c),r) & !at_start(c) & moving(c)
  => [(min (length(loc(c))/120)*60, 0.5), INF]
 loc(c) = r, at_start(c) = T.

u_turn {c : is_car(c)}:
 moving(c) & !(is_any opposite(loc(c))) => [2,INF]
 loc(c) = opposite(loc(c)), at_start(c) = F.

stop {c : is_car(c)}: moving(c) => [0,INF] moving(c) = F.

# Constraints:
```

```
stay_in_road {c, r : is_car(c) & is_link(r)}:
 loc(c) = r & moving(c) : (length(r)/1.8)*60.

# Expectations:

# alpha: no turning back
expect A{c : is_car(c)}
 A[0,INF] let r0 = loc(c) in
  X[0,INF] (loc(c) = r0 | (starts_at(loc(c)) = ends_at(r0) &
                           !loc(c) = opposite(r0))).

# beta: minimal and maximal time in road
expect A{c : is_car(c)}
 A[0,INF] let r0 = loc(c) in
  (X[0,INF]
   ((loc(c) = r0) | let r1 = loc(c) in
    A[0,(length(loc(c))/(avg_speed(loc(c))+15))*60] loc(c) = r1) &
   E[0,(length(loc(c))/(avg_speed(loc(c))-5))*60]
    (!(loc(c) = r0) | !moving(c))).

# gamma: don't stop except at destination
expect A{c : is_car(c)}
 A[0,INF] (ends_at(loc(c)) = dest(c) | moving(c)).

# delta: shortest path (almost) to destination
expect A{c : is_car(c)}
 A[0,INF]
 ((length(loc(c))/avg_speed(loc(c))) +
  distance(ends_at(loc(c)),dest(c))) <=
 min {r1 : is_link(r1) & starts_at(r1) = starts_at(loc(c))}
  ((length(r1)/avg_speed(r1)) + distance(ends_at(r1),dest(c)))
  + 2.
```

B.2 A Sample Scenario

The following is a sample scenario defined for the model.

```
# There is only one car.
object c1.
car(c1) = T.

# Initially, the car is at the beginning of link 232 and moving.
# The destination is unknown, but restricted to be a terminal node.
```

```
init assign loc(c1) = l232, at_start(c1) = T, moving(c1) = T.
init select is_node(dest(c1)) & terminal(dest(c1)).

# Observation: three and a half minutes later, the car is in link
# 332.
obs [3.5] loc(c1) = l332.

# Queries:
P[]  A[2.0,5.0]loc(c1) = r332? # is it possible that the car remains
                               # in link 332 throughout [2.0,5.0]?
N[]  A[3.0,4.0]loc(c1) = r332? # must the car necessarily remain in
                               # link 332 throughout [3.0,4.0]?
P[1] E[3.5,5.5]loc(c1) = r306? # is it possible for the car to be in
                               # link 306 at some point in [3.5,5.5],
                               # if all expectations except alpha are
                               # ignored?

W[] [5.5]loc(c1)?  # what are the possible location of the car 2 min
                   # after it was observed in link 332?
W[] [4]dest(c1)?   # what are the destination consistent with both
                   # the initial and later observations?
```

References

Alur, R., and Dill, D. 1994. A theory of timed automata. *Theoretical Computer Science* 126(2):183 – 236.

Alur, R., and Henzinger, T. 1994. A really temporal logic. *Journal of the ACM* 41(1):181 – 204.

Alur, R.; Henzinger, T.; Lafferiere, G.; and Pappas, G. 2000. Discrete abstractions of hybrid systems. *Proceedings of the IEEE* 88(7):971 – 984.

Alur, R.; Courcoubetis, C.; and Dill, D. 1993. Model checking in dense real-time. *Information and Computation* 104(1):2 – 34.

Alur, R.; Feder, T.; and Henzinger, T. 1996. The benefits of relaxing punctuality. *Journal of the ACM* 43(1):116 – 146.

Alur, R.; Henzinger, T.; and Ho, P. 2000. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering* 22(3):181 – 201.

Alur, R. 1999. Timed automata. In *NATO-ASI Summer School on Verification of Digital and Hybrid Systems.* Available at `http://www.cis.upenn.edu/~alur/Nato97.ps.gz`.

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and Computation* 75(2):87 – 106.

Bacchus, F., and Kabanza, F. 1996. Planning for temporally extended goals. In *Proc. 13th National Conference on Artificial Intelligence (AAAI'96).*

Beetz, M., and Grosskreutz, H. 2000. Probabilistic hybrid action models for predicting concurrent percept-driven robot behaviour. In *Proc. 5th International Conference on AI Planning Systems*, 42 – 51.

Beetz, M., and McDermott, D. 1992. Declarative goals in reactive plans. In *Proc. 1st International Conference on AI Planning Systems.*

Ben Lamine, K., and Kabanza, F. 2000. History checking of temporal fuzzy logic formulas for monitoring behavior-based mobile robots. In *Proc. IEEE International Conference on Tools with Artificial Intelligence.*

Blum, A., and Furst, M. 1997. Fast planning through graph analysis. *Artificial Intelligence* 90(1–2):281 – 300.

Blythe, J. 1998. *Planning under Uncertainty in Dynamic Domains.* Ph.D. Dissertation, Carnegie Mellon University.

Blythe, J. 1999. Decision-theoretic planning. *AI Magazine* 20(2).

Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proc. 5th International Conference on Artificial Intelligence Planning and Scheduling.*

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1 – 94.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32:333 – 377.

Chatfield, C. 1996. *The Analysis of Time Series – An Introduction.* Chapman & Hall, 5th edition.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proc. 15th National Conference on Artificial Intelligence (AAAI'98).*

Clarke, E.; Emerson, E.; and Sistla, A. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8(2):244 – 263.

Clarke, E.; Grumberg, O.; and Peled, D. 1999. *Model Checking.* MIT Press.

Collins, G., and Pryor, L. 1995. Planning under uncertanity: Some key issues. In *Proc. 14th International Joint Conference on Artificial Intelligence.*

Coradeschi, S., and Saffiotti, A. 2000. Anchoring symbols to sensor data: Preliminary report. In *Proc. 17th National Conference on Artificial Intelligence.*

Coradeschi, S.; Karlsson, L.; and Nordberg, K. 1999. Integration of vision and decision-making in an autonomous airborne vehicle for traffic surveillance. In *Computer Vision Systems*, volume 1542 of *Lecture Notes in Computer Science.* Springer Verlag. 216 – 230.

Cowel, R.; Dawid, A.; Lauritzen, S.; and Spiegelhalter, D. 1999. *Probabilistic Networks and Expert Systems.* Springer Verlag.

Davis, R.; Shrobe, H.; and Szolovits, P. 1993. What is a knowledge representation? *AI Magazine* 14(1):17 – 33.

de Kleer, J., and Williams, B. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32:97 – 130.

Dean, T., and Wellman, M. 1991. *Planning and Control.* Morgan Kaufmann.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61 – 95.

DeCoste, D. 1990. Dynamic across-time measurement interpretation. In *Proc. 8th National Conference on Artificial Intelligence (AAAI'90).*

Despouys, O., and Ingrand, F. 1999. Propice-Plan: Toward a unified framework for planning and execution. In *Proc. 5th European Conference on Planning (ECP'99).*

Despouys, O. 2000. *Une Architecture Intégrée pour la Planification et le Contrôle d'Exécution en Environnement Dynamique.* Ph.D. Dissertation, l'Institut National Polytechnique de Toulouse.

Doherty, P.; Granlund, G.; Kuchcinski, K.; Sandewall, E.; Nordberg, K.; Skarman, E.; and Wiklund, J. 2000. The WITAS unmanned aerial vehicle project. In *Proc. European Conference on Artificial Intelligence.*

Emerson, E. 1990. Temporal and modal logic. In *Handbook of Theoretical Computer Science.* Elsevier. 997 – 1072.

Florian, M., and Nguyen, S. 1976. An application and validation of equilibrium trip assignment methods. *Transportation Science* 10(4):374 – 390.

Forbes, J.; Huang, T.; Kanazawa, K.; and Russell, S. 1995. The BATmobile: Towards a Bayesian automate taxi. In *Proc. 14th International Joint Conference on Artificial Intelligence.*

Forbus, K. 1986. Interpreting measurements in physical systems. In *Proc. 5th National Conference on Artificial Intelligence (AAAI'86).*

Garcia, C.; Prett, D.; and Morari, M. 1989. Model predictive control: Theory and practice – a survey. *Automatica* 25(3):335 – 348.

Gerth, R.; Peled, D.; Vardi, M.; and Wolper, P. 1995. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the 15th Workshop on Protocol Specification, Testing and Verification.* North-Holland.

Ghallab, M. 1996. On chronicles: Representation, on-line recognition and learning. In *Proc. 5th Internation Conference on Principles of Knowledge Representation and Reasoning*, 597 – 607. Morgan Kaufmann.

Godefroid, P., and Wolper, P. 1993. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design* 2(2):149 – 164.

Goldman, R.; Musliner, D.; Krebsbach, K.; and Boddy, M. 1997. Dynamic abstraction planning. In *Proc. 14th National Conference on Artificial Intelligence (AAAI'97).*

Gordon, N.; Salmond, D.; and Smith, A. 1993. A novel approach to non-linear and non-gaussian bayesian state estimation. *IEEE Proceedings* 140:107 – 113.

Grewal, M., and Andrews, A. 1993. *Kalman Filtering: Theory and Practice.* Prentice Hall.

Haslum, P. 2002. Partial state progression: An extension to the Bacchus-Kabanza algorithm, with applications to prediction and MITL consistency. In *Proc. AIPS 2002 workshop on Planning via Model Checking.*

Hayes, P. 1979. The logic of frames. In *Frame Conception and Text Understanding.* Walter de Gruyter and Co. 46 – 61. Reprinted in Brachman, R.J. and Levesque, H.J. (eds.) "Readings in Knowledge Representation". Morgan Kaufmann. 1995.

Hoel, P.; Port, S.; and Stone, C. 1972. *Introduction to stochastic processes.* Houghton Mifflin.

Hoel, P. 1984. *Introduction to Mathematical Statistics.* Wiley.

Huang, T., and Russell, S. 1997. Object identification in a bayesian context. In *Proc. 15th International Joint Conference on Artificial Intelligence.*

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99 – 134.

Kalman, R. 1960. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering.*

Kearns, M., and Valiant, L. 1989. Cryptographic limitations on learning Boolean formulae and finite automata. In *Proc. 21st Annual ACM Symposium on Theory of Computing*, 433 – 444. Association for Computing Machinery.

Kitagawa, G. 1996. Monte carlo filter and smoother for non-gaussian non-linear state space models. *Journal of Computational and Graphical Statistics* 5:1 – 25.

Koenig, S., and Simmons, R. 1995. Real-time search in non-deterministic domains. In *Proc. 14th International Joint Conference on Artificial Intelligence.*

Kowalski, R., and Sergot, M. 1986. A logic-based calculus of events. *New Generation Computing* 4.

Ljung, L. 1987. *System Identification: Theory for the User.* Prentice-Hall.

Matstoms, P.; Jönsson, H.; and Carlsson, A. 1996. Beräkning av volume/delay-funktioner för nätverksanalys. *VTI Meddelande* 777.

Maybeck, P. 1979. *Stochastic Models, Estimation and Control*, volume 1. Academic Press.

McDermott, D. 1994. An algorithm for probabilistic totally-ordered temporal projection. Technical Report YALEU/CSD/RR 941, Yale University.

McIlraith, S.; Biswas, G.; Clancy, C.; and Gupta, V. 2000. Hybrid systems diagnosis. In *Proc. International Workshop on Hybrid Systems: Computation and Control.*

McIlraith, S. 1998. Explanatory diagnosis: Conjecturing actions to explain observations. In *Proc. 6th International Conference on Principles of Knowledge Representation and Reasoning.*

Mesarovic, M., and Takahara, Y. 1975. *General Systems Theory: Mathematical Foundations.* Academic Press.

Morse, P. 1978. Search theory. In Moder, J., and Elmaghraby, S., eds., *Handbook of Operations Research.* van Nostrand Reinhold. chapter III-6.

Musliner, D.; Durfee, E.; and Shin, K. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74(1):83 – 127.

Pasula, H.; Russell, S.; Ostland, M.; and Ritov, Y. 1999. Tracking many objects with many sensors. In *Proc. 16th International Joint Conference on Artificial Intelligence.*

Patriksson, M. 1994. *The Traffic Assignment Problem: Models and Methods.* Utrecht, The Netherlands: VSP.

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann.

Peot, M., and Smith, D. 1992. Conditional nonlinear planning. In *Artificial Intelligence Planning Systems: Proc. International Confrence.*

Pitt, M., and Shephard, N. 1999. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association* 94(446):590 – 599.

Pitt, L. 1989. Inductive inference, DFAs and computational complexity. In *Proc. of the 1989 International Workshop on Analogical and Inductive Inference*, volume 397 of *Lecture Notes in Computer Science.* Springer Verlag.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley.

Qin, S., and Badgwell, T. 1997. An overview of industrial model predictive control technology. In *Chemical Process Control – V*, volume 93 of *AIChE Symp. Series.* American Institute of Chemical Engineers.

Rosen, R. 1985. *Anticipatory Systems.* Pergamon Press.

Rosen, R. 1991. *Life Itself: A Comprehensive Inquiry into the Nature, Origin, and Fabrication of Life.* Columbia University Press.

Saffiotti, A. 1998. *Autonomous Robot Navigation: A Fuzzy Logic Approach.* PhD Thesis, Faculté de Science Appliquées, IRIDIA, Université Libre de Bruxelles.

Schwendimann, S. 1998. A new one-pass tableau calculus for PLTL. In de Swart, H., ed., *Proc. International Conference on Theorem Proving with Analytic Tableaux and Related Methods*, volume 1397 of *Lecture Notes in Artificial Intelligence*, 277 – 291. Springer Verlag.

Smith, D., and Weld, D. 1998. Conformant Graphplan. In *Proc. 15th National Conference on Artifical Intelligence (AAAI'98).*

Thiébaux, S., and Lamb, P. 2000. Combining Kalman filtering and Markov localization in network-like environments. In *Proc. 6th Pacific-Rim International Conference on Artificial Intelligence.*

Thrun, S. 2000. Probabilistic algorithms in robotics. *AI Magazine* 21.

Tijms, H. 1994. *Stochastic Models.* Wiley.

Tversky, A., and Kahneman, D. 1974. Judgement under uncertainty: Heuristics and biases. *Science* 185:1124 – 1131.

Washington, R. 2000. On-board real-time state and fault identification for rovers. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'00).*

Weld, D.; Anderson, C.; and Smith, D. 1998. Extending Graphplan to handle uncertainty & sensing actions. In *Proc. 15th National Conference on Artifical Intelligence (AAAI'98).*

Williams, B., and Nayak, P. 1996. A model-based approach to reactive self-configuring systems. In *Proc. 13th National Conference on Artificial Intelligence*.

Yin, R. 1994. *Case Study Research: Design and Methods*. Sage.