SPATIAL REPRESENTATION, REASONING AND CONTROL FOR A SURVEILLANCE SYSTEM
THE UNIVERSITY OF LONDON

SPATIAL REPRESENTATION, REASONING AND CONTROL FOR A SURVEILLANCE SYSTEM

RICHARD JOHN HOWARTH
QUEEN MARY AND WESTFIELD COLLEGE

SUBMITTED APRIL 1994, MINOR REVISIONS JULY 1994

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF LONDON IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
ABSTRACT

The aim of the work reported in this dissertation is the development of a computational understanding of the surveillance of moving objects and their interactions in real world situations. Understanding the activity of moving objects starts by tracking objects in an image sequence, but this is just the beginning. The objective of this work is to go further and form conceptual descriptions that capture the dynamic interactions of objects in a meaningful way. The computational approach uses the results from a low- and intermediate-level vision system developed elsewhere. The issues concerned with extending computational vision to address high-level vision are described in the context of a surveillance system. These issues are separated into three main concerns: contextual-indexing which is the representation of the contextual information present in the environment through which the moving objects travel; event-reasoning which is the identification of behavioural primitives, their selection and composition; and task-based-control which is guiding system performance to comply with a given surveillance task.

Contextual-indexing involves accessing pre-compiled information concerning typical object behaviour in relation to the environment to provide contextual cues for the formation of conceptual descriptions. The spatial information is held in a hierarchical database to represent overlapping spatial contexts. A computer program called SPATIAL-LAYOUT implements this database.

Event-reasoning involves building temporally evolving episodic descriptions that act as basic elements for conceptual descriptions. The identification of conceptual descriptions can be pipelined to provide a database of results about the activities of all the moving objects, and which can be used to support query based reasoning. A computer program called HIVIS-MONITOR implements this capability.

Task-based-control involves identifying those scene objects likely to be worth attending because they fulfill the given surveillance task. In contrast to the query based approach we no longer collect information about all the scene objects, instead only data that is potentially task related is processed. A computer program called HIVIS-WATCHER implements this kind of attentional approach.

Together these three components provide a solution to the surveillance problem.
CONTENTS

Abstract ii

Acknowledgements viii

Chapter 1 Introduction 1
1 Reasoning about what we see 2
2 A surveillance system 5
3 A reader’s guide 9

Chapter 2 Knowledge in image understanding 10
1 Categories of knowledge 11
2 Spatial representation 13
  2.1 Surveillance 13
  2.2 Computational approaches 13
  2.3 A common representation 20
  2.4 Polyhedral tessellations and implementations 21
  2.5 Summary 24
3 Events and behaviour 25
  3.1 Surveillance 25
  3.2 Computational approaches 25
  3.3 Temporal logic 34
  3.4 Tracking 35
  3.5 Summary 37
4 Control and planning 38
  4.1 Surveillance 38
  4.2 Observation and the environment 39
  4.3 Computational approaches 43
  4.4 Summary 49
5 Conclusions 51

Chapter 3 Spatial representation 52
1 Cell space 53
2 Hierarchical database 54
  2.1 Leaf and composite regions 54
  2.2 Reasoning with regions 56
  2.3 Contextual indexing 57
| Contents |
|----------|-------|
| 6.5 Summary | 128   |
| 7 Situations and routines | 129   |
| 7.1 Routines | 129   |
| 7.2 Examples | 130   |
| 7.3 Summary | 133   |
| 8 Conclusions | 134   |

Chapter 5 Attentional control | 136   |
| 1 The perceiver | 137   |
| 1.1 Basic architecture | 138   |
| 1.2 Extending the central-system | 139   |
| 2 Virtual world | 141   |
| 2.1 Buffer | 141   |
| 2.2 Perceived environment | 141   |
| 2.3 Display | 142   |
| 3 Scene agents | 143   |
| 3.1 Typical object model | 143   |
| 3.2 Agency | 144   |
| 3.3 Example | 145   |
| 3.4 Summary | 146   |
| 4 Task based control | 147   |
| 4.1 Guiding computation | 148   |
| 4.2 Allocate | 148   |
| 4.3 Collect | 149   |
| 4.4 Summary | 149   |
| 5 Implementation and formalism details | 150   |
| 5.1 Framework | 150   |
| 5.2 Dynamic Decision Networks | 150   |
| 5.3 Tasknet | 157   |
| 5.4 Mechanism | 159   |
| 5.5 Summary | 159   |
| 6 Proximity | 161   |
| 6.1 Selecting interesting features | 161   |
| 6.2 Meta level | 165   |
| 6.3 Examples | 167   |
| 6.4 Summary | 173   |
| 7 Kernel | 175   |
| 7.1 Perceived environment | 175   |
| 7.2 Markers and operators | 175   |
| 7.3 Global form | 176   |
| 7.4 Implementation details | 177   |
| 7.5 Summary | 177   |
# Contents

8 Coordination ............................................. 178  
  8.1 Gross change in motion .............................. 178  
  8.2 Giveaway ............................................. 180  
  8.3 Summary ............................................... 188  
9 Grouping .................................................. 190  
  9.1 Clusters ............................................... 190  
  9.2 Overtaking ........................................... 191  
  9.3 Queueing .............................................. 197  
  9.4 Summary ............................................... 197  
10 Conclusions .............................................. 198  
  10.1 Elements ............................................ 198  
  10.2 Evaluation ......................................... 199  
  10.3 Extensions ......................................... 201  

Chapter 6  Conclusion ......................................... 203  
1 Recapitulation ........................................... 204  
2 Contributions ............................................. 206  
  2.1 Spatial representation ............................. 206  
  2.2 Events and behaviour ............................... 207  
  2.3 Control and planning ............................... 208  
3 Future work ............................................... 209  
  3.1 Learning intrinsic environmental properties ...... 210  
  3.2 Alternatives to representing the world .......... 211  
  3.3 A unified model .................................... 211  
4 Epilogue .................................................. 212  

Appendix A  Video footage and scenarios .................. 213  

Appendix B  Mathematical development ...................... 224  
1 Background for the common representation ............. 225  
  1.1 Topology ............................................. 225  
  1.2 Simplicial Topology ................................ 227  
  1.3 Cellular topology .................................. 228  
  1.4 Adjacency sets and incidence structure ......... 230  
  1.5 Inter region boundaries ........................... 231  
2 Background for regions .................................. 233  
  2.1 The skeleton ........................................ 233  
  2.2 The subcomplex .................................... 233  
  2.3 The subdivision .................................... 234  
  2.4 Tessellation techniques ............................ 234  
3 Database primitives ..................................... 236  
  3.1 Leaf split .......................................... 236  
  3.2 Leaf merge ......................................... 236
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3 Composite union</td>
<td>236</td>
</tr>
<tr>
<td>4 Example spatial layout file</td>
<td>238</td>
</tr>
<tr>
<td>5 Cell parser</td>
<td>239</td>
</tr>
<tr>
<td>6 Tolerance space</td>
<td>241</td>
</tr>
<tr>
<td>6.1 Camera view</td>
<td>241</td>
</tr>
<tr>
<td>6.2 Objects</td>
<td>242</td>
</tr>
<tr>
<td>Appendix C Spatio-temporal mechanisms</td>
<td>243</td>
</tr>
<tr>
<td>1 Time</td>
<td>244</td>
</tr>
<tr>
<td>1.1 Representation issues</td>
<td>244</td>
</tr>
<tr>
<td>1.2 Cellwise-time language</td>
<td>248</td>
</tr>
<tr>
<td>1.3 Intervals</td>
<td>253</td>
</tr>
<tr>
<td>2 Space and time</td>
<td>255</td>
</tr>
<tr>
<td>2.1 Interval tree</td>
<td>255</td>
</tr>
<tr>
<td>2.2 Modelling paths</td>
<td>257</td>
</tr>
<tr>
<td>3 Spatial arrangements</td>
<td>263</td>
</tr>
<tr>
<td>3.1 Local object locations</td>
<td>263</td>
</tr>
<tr>
<td>3.2 Proximity</td>
<td>265</td>
</tr>
<tr>
<td>Appendix D Bayesian Networks</td>
<td>266</td>
</tr>
<tr>
<td>1 Causal polytrees</td>
<td>267</td>
</tr>
<tr>
<td>1.1 Computational approach</td>
<td>267</td>
</tr>
<tr>
<td>1.2 Implementation details</td>
<td>270</td>
</tr>
<tr>
<td>2 Weights</td>
<td>272</td>
</tr>
<tr>
<td>Appendix E Operational semantics for MACNET</td>
<td>274</td>
</tr>
<tr>
<td>1 Gates</td>
<td>276</td>
</tr>
<tr>
<td>2 Gate language details</td>
<td>280</td>
</tr>
<tr>
<td>3 MACNET</td>
<td>290</td>
</tr>
<tr>
<td>3.1 Language components</td>
<td>292</td>
</tr>
<tr>
<td>3.2 Extensions</td>
<td>294</td>
</tr>
<tr>
<td>4 MACNET language details</td>
<td>296</td>
</tr>
<tr>
<td>5 Summary</td>
<td>304</td>
</tr>
<tr>
<td>Bibliography</td>
<td>307</td>
</tr>
<tr>
<td>Index</td>
<td>323</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

First, I wish to thank my advisor, Hilary Buxton, for her continuous support and encour-
age. Even after her move to Sussex University, Hilary read through the various
drafts, and I always returned from the trips to Brighton with renewed purpose and doubts
resolved. At QMW, Mike Clarke, and then Dave Saunders, after Mike became ill, took
over the supervision of my work. Both of whom have listened to early ideas and discussed
their validity.

Work on this thesis began as part-time postgraduate research while I was employed
by GEC-Marconi and working on the ESPRIT II project 2152, VIEWS. I am grateful to
Kevin Poulter who made it possible for me to start the work reported here, Graham Hill
for being the best of technical managers and Tony Clark for developing my appreciation
of Lisp and functional programming. I also thank the members of the VIEWS project,
their work provided the background for what has been developed here.

After two years part-time, I transferred to full-time status and was funded by Science
and Engineering Research Council under a CASE award with the GEC Marconi Research
Centre. Ken Paler was most helpful in arranging the CASE award.

It has been a pleasure to complete this dissertation at the Department of Computer
Science, the members of which, particularly those in the Vision Laboratory, have helped in
indirect but important ways. I also thank Susan Richards for furnishing all the wonderful
inter-library loans, and Jon Hill, my collaborator in the ongoing battle with \LaTeX{}.

I am grateful for discussions with John Bell, Bernard Buxton, Margaret Fleck, Sha-
gang Gong, Peter Landin, Andrew Toal, and Edward Tsang. They also provided advice
and insightful comments on drafts of this dissertation, or the papers that it has been based
on.

Finally, I thank my parents for their dedicated support of my academic pursuits.
Chapter 1

Introduction

This dissertation describes a computational approach to high-level vision and its implementation in two HIVIS (High-level VISion) systems. We discuss how Artificial Intelligence (AI) techniques involving spatial reasoning can be applied to the problem of surveillance. In the surveillance problem the perceiver plays a passive role, not physically interacting with its environment. However, even without this ability to interact directly with the environment by doing things like asking questions, touching, and moving around, we would expect the perceiver to understand the activity unfolding in the scene being observed. We restrict this general surveillance problem by not considering further how the perceiver learnt about the structural properties of the scene, instead we will consider the case where we have a partially known scene with known object types.

The surveillance of objects in a partially known scene is the process by which a description of the environment, in terms of objects and known ground-plane, enables the perceiver to evolve an understanding of what is happening in the scene. For example if we placed a camera at the side of a roundabout we would require various environmental details of the scene layout and objects that form the road-traffic. The aim of the study described here is to develop a computational approach to this process. This computational theory includes two parts: (1) the requirements and principles of the computation, and (2) the mechanisms that support it. This separation of computation and mechanism is to make clear the independence of the computation performed at the functional level from the physical mechanism that supports the computation. The computational investigation concerns the nature of the internal representations used and the processes by which they are derived. The study of the mechanisms concerns how the computational representations and processes can be implemented.

We choose an approach based on spatial representation and reasoning as we can directly perceive spatial properties. We model time indirectly in terms of changes in these properties.
1 **Reasoning about what we see**

In the modern world there is an increasing use of surveillance, resulting in the need for automatic or semi-automatic methods for processing the resulting dynamic input data. Surveillance concerns more than just observation, in addition to having some intelligence and knowledge the perception performed is much more purposive, complying to some known task. We call the identification of a computational theory that includes provision for intelligence, knowledge and task directedness the "surveillance problem".

To perform surveillance we need to reason about the activities of the objects that are perceived. The process by which this visual perception is performed is complex and will not be fully addressed here. However, since perception is an important part of surveillance we need to sketch the relationship of the work described here to the framework of computational vision. As illustrated in figure 1.1 we can separate vision into three stages: low-level (or early), intermediate-level and high-level (or late). Low-level vision is the best understood (Horn [103], Marr [150]), and concerns visual receptors, be they from a television camera (basically just producing a 2D array of grey level intensity values) or biological, with low-level processing using visual primitives that act on the results from the visual receptors, to provide image features such as edges, corners and flow vectors. Intermediate vision is less well understood, and concerns the recognition of objects (e.g., model matching and tracking). Marr describes a framework for 3D interpretation, that begins to address intermediate-level vision. High-level vision is the least well understood and concerns the interpretation of the evolving information that is provided by intermediate-level vision as well as directing what intermediate-level visual processing should be performed. In progressing up these levels we see that image oriented information is at the lower levels and the more abstract, symbolic descriptions are at the higher levels. It is the development of high-level visual processing that allows the results from intermediate-level visual processing to be used for reasoning over longer time scales to obtain a greater understanding of what is going on in the field-of-view. In this dissertation we will concentrate on the role of high-level vision. Emphasis will be placed upon how what we know about an environment affects the interpretation of the observed object behaviour.
In the surveillance problem we will be taking the viewpoint of a single static perceiver observing a wide-area scene. This wide field-of-view is to allow object activity to be played out in front of us. The single static perceiver, a fixed camera, ignores difficult issues associated with active cameras and multisensor fusion, while providing ample visual data to address surveillance tasks. By concentrating on the role of high-level vision we are saying in effect that the discussion of reasoning about what we see begins once we have recognized the object/s we are looking at. This stance, although feasible in some situations, brings with it a bottom-up flow of information from images to high-level vision that ignores the fact that not all the original image data may be required all of the time.

Figure 1.1 shows where the interface falls between intermediate- and high-level processing. This interface was initially for the pragmatic purpose of enabling research to be performed. Drawing the interface in this position allows the results from intermediate-level processing to be collected, as a stream of compact encodings for subsequent high-level processing. The problems ensuing from this initial decision are highlighted in the final stages of the research reported here where a more active approach to high-level vision is developed.

As illustrated in figure 1.2 a better, and more realistic, approach is to consider how high-level vision can use its accumulated recent knowledge to guide the intermediate-level processing which in turn selects the necessary low-level vision data. This top-down control results in a more focussed flow-of-information back up to high-level vision which should be more appropriate, and the feedback would provide a tight coupling between intermediate- and high-level processing. Unfortunately moving the interface, shown in figure 1.1, back one stage would require addressing much more material than is possible within the scope of a single thesis. However, such developments are seen as essential future work for an ideal architecture.

We are addressing the surveillance problem with simplifying assumptions:

**Assumption 1** The computer programs are just prototypes that are not yet required to operate in real world applications.

**Assumption 2** We are using a known 2D ground-plane.

**Assumption 3** The ground-plane is static.

**Assumption 4** We are given a stream of compact-encodings that represent the output from intermediate-level processing.
Assumption 5 The official-observer has a viewpoint from a single fixed camera.

Assumption 6 The surveillance task is understanding the activity of moving objects in the scene.

Assumption 7 Reasoning in accurate time is not required.

Assumption 8 The description of system behaviour and the behaviour of observed objects can be defined by the program designer.

In addressing the surveillance problem the first stage we consider, called spatial representation, is the representation of the environment in terms of the scene that is visible to our official observer. Once we have this spatial foundation in place, we are then in a position to address issues associated with dynamic environmental properties, namely the movements of scene objects. In this second stage, called events and behaviour, we are interpreting the results from intermediate-level processing to identify relevant symbolic abstractions. Pioneering work by Nagel [171] and Neumann [173] has addressed this issue of how to form a conceptual description from image sequences. We extend this in the third stage, called control and planning, by addressing task directedness. It is by integrating these three stages that we are able to fully address the surveillance problem.

In the surveillance problem the official-observer is not a partner in the interaction taking place in the scene, and although the official-observer visually attends to them, the observed people/objects\(^1\) are not necessarily aware of this. The things that are relevant to the official-observer are likely to differ greatly from those of the parties involved in the interaction. This permits the official-observer to see at the same time more and less than what is seen by the participants, with only the fragments of action manifested by them being accessible for observation. To understand these fragments, the official-observer has to use knowledge of similar patterns of interaction with the goal of constructing the motives of the actors from that part of its knowledge pertinent to the observation. The knowledge used by the official-observer is therefore different to that used by the participants in the interaction. Even with this difference there is a chance, sufficient for most practical purposes, that the subjective meaning of the actor's acts can be understood. This chance increases with the degree of anonymity and standardisation present in the observed behaviour. From this it should be clear that surveillance is a difficult problem, and to make it more tractable we mainly consider application domains that contain a high degree of standardisation and structure.

\(^1\)We use the word "object" to refer to physical entities like cars, trams, and planes, where the person operating the machine may not be visible to the official-observer.
2 A surveillance system

Having so far considered the general issues of the surveillance problem, in this section we will briefly describe the ESPRIT II project 2152, VIEWS (Visual Inspection and Evaluation of Wide-area Scenes). The VIEWS project provides a concrete example of a working system that can provide the intermediate-level vision results required by HIVIS.

An outline of the basic architecture for the project is given in figure 1.3, showing how it’s functionality is divided into the Perception Component (PC) for low- and intermediate-level vision and the Situation Assessment Component (SAC) for high-level vision. The project has developed an advanced vision system (Corrall and Hill [46]) for surveillance of uncontrolled, outdoor, dynamic scenes in a partially known environment. Application domains include: a holding area at Heathrow airport, in which aeroplanes manoeuvre prior to take-off; stand area monitoring at Newcastle airport, where service vehicles approach to load fuel and catering supplies; and a German roundabout which is used by vehicles and pedestrians that travel through it on route to their respective destinations. In this dissertation we will draw examples from the road application and another example of an indoor room. An initial example is shown in figure 1.4 which shows three image frames selected from a sequence taken at the German roundabout. In this part of the sequence a number of episodic behaviours are unfolding: one vehicle leaves the roundabout; another is in an entry lane to the roundabout; also towards the rear of the image a car begins to overtake a lorry.

The PC is a combination tracker and 3D model matcher (Worrall et al. [259]) which supplies the SAC with an asynchronous stream of vehicle “data packets”. Each data packet describes: classification, estimated velocity, orientation, vehicle outline (ground-plane projected posebox). Example data packets from the three frames used in figure 1.4 are given in figure 1.5\(^2\). As shown by the data packets in figure 1.5 the interface between

\(^2\)The PC component has available accurate velocity and orientation data which are calculated as part of the low-level tracking processes. Classification, on the other hand is a much more complex issue. In the data set here, the velocity and orientation is implicit in the 2D pose.
Figure 1.4: Three images showing typical vehicle activity on the roundabout and how the 3D pose descriptions can be transformed to a ground-plane view.
Figure 1.5: Three frames of processed image data in the form of data packets called "compact encodings" as they only represent the poseboxes of moving objects.
the PC and the SAC is primarily built upon the results from the model matcher which we refer to as the stream of "compact encodings".

VIEWS has been demonstrated on data from actual airside ground movements that occur during stand area servicing as observed by a single camera. When performing ground movements surveillance of apron areas it is necessary to locate and deploy servicing vehicles for efficient aircraft turnaround. Vehicles of many different types visit the aircraft in order to provide specific servicing functions such as refuelling, emptying and loading of baggage, and so on. The area round the plane becomes cluttered with many vehicles and it is not difficult to lose track of what servicing has to be done. The requirement is to be able to detect and classify vehicles, to determine what serving has been carried out, what servicing is yet to come, and from that, to be able to predict when the aircraft will be ready. Results from this scenario are described in Corrall and Hill [46].

VIEWS has also been demonstrated on data from a German roundabout called "Bremer Stern", an example of which is given in figure 1.4. The requirement is to detect and partially classify vehicles and correlate their relevance to the user's surveillance task. This surveillance task may just involve collecting statistics, such as the number and type of vehicle that use different areas of the scene, or identification of the number of vehicles that use a particular route through the scene. More complex surveillance tasks involve the detection of "incidents" where some significant deviation from normal behaviour is identified. Corrall and Hill discuss example surveillance tasks for motorway and urban traffic domains. Results showing how VIEWS operates on the Bremer Stern scenario are not given in [46], although this is recognised as a potential application domain.

There is a market for specific applications of computer vision systems capable of performing surveillance and, although the work described in this dissertation is a step in the right direction, additional development is needed before a generic but tailored product capable of fulfilling this market is available.
3 A reader's guide

This introduction has described the importance and need for automated surveillance. The dissertation continues with a chapter covering background material, picking out the relevant foundations from related work. Chapter 2 has three areas concerning spatial representation, events and behaviour, and control and planning, which are not often brought together. We continue these three strands, giving each its own chapter.

Chapter 3 is about spatial representation, providing the spatial foundations used in following chapters by covering details of how space is represented using cells and how these cells can be stored in a hierarchical database called SPATIAL-LAYOUT. We develop the mathematics underlying this approach separately in appendix B. Chapter 3 is completed by a description of a computer program that implements SPATIAL-LAYOUT and a road-traffic domain example illustrating its use.

Once the spatial foundations of the ground-plane are in place, we follow on in chapter 4 with events and behaviour. This chapter begins by introducing primitives that are common to both HIVIS systems described here. We then consider how events and episodes can be identified from these primitives, and develop the mechanisms that support this approach in a computer program called HIVIS-MONITOR. In appendix C we describe details of the spatial mechanism. After illustrating HIVIS-MONITOR's performance with some examples drawn from the road-traffic domain data sets given in in appendix A, we reassess this approach. In the rest of chapter 4 we begin to address the problems raised by HIVIS-MONITOR's inability to describe the spatial arrangements between two objects. Addressing these failings of HIVIS-MONITOR helps identify further requirements of control and planning present in the surveillance problem.

In chapter 5 we address control and planning, which results in the development of a new system called HIVIS-WATCHER that draws upon previous work on "situated activity" (Agre and Chapman [3]) and attentional control (Ballard [14]). We describe the MACNET language of Agre and Chapman in appendix E and provide additional details of the Bayesian approach taken in appendix D.

Finally, in chapter 6, we summarise the main contributions, drawing together results from its preceding three chapters and discuss some open problems.
CHAPTER 2

KNOWLEDGE IN IMAGE UNDERSTANDING

Before addressing the development of surveillance systems, we first identify useful foundations from previous work. These foundations provide approaches to interpreting the available data. For example, it is by using additional knowledge of the task and scene that it becomes feasible to understand processed image data. The progression through the survey corresponds to an investigation of how spatial representation and then reasoning about the data contained in that representation, can best be performed in the context of the surveillance problem. Issues connected with spatial reasoning are addressed in relation to understanding the activity of agents in an environment, and it is consideration of these issues about events and behaviour that show the necessity of control and planning.

We discuss these three areas after introducing a four layer model to describe their interdependencies. We need a synthesis of spatial representation, events and behaviour, and control and planning, to address the surveillance problem.
1 Categories of knowledge

There are three areas of interest covered in this chapter which correspond to the three methodologies of our computational approach. The methodologies are (1) spatial representation, (2) events and behaviour, and (3) control and planning. These areas of interest do not fit easily into one unifying framework, however, they do have an operational ordering and interdependencies. To explain the operational ordering we need to introduce some terms that express a four layer model categorising the representation and reasoning present in HIVIS. This model is based upon part of the KADS approach to expressing expert knowledge [254], and the terms are as follows:

- The domain layer contains domain concepts, attributes, relations and facts.
- The inference layer defines the relationships arising in a task context and performs inferences on knowledge defined at the domain layer. The possible inferences are determined by the available data at the domain layer.
- The task layer describes how a task is decomposed to control the necessary inferences. Each task is completely described by its goal, and its performance is dependent upon which inference layer rules are satisfied.
- The strategic layer defines the goals from the task layer that are relevant to the current problem or situation.

These four layers are used to describe the three methodologies outlined above in the following way:

- The spatial representation methodology includes domain layer concepts and the particular application-domain data that together express the geometric structure of the environment. It also includes inference layer functions for manipulating and structuring the spatial concepts. Most of the representation concerns static data, although the dynamic scene objects also have important spatial properties.

- The events and behaviour methodology also encompasses the domain and inference layers. In the domain layer, concepts and application-domain data express the typical object behaviour for the scene, with the inference-layer holding the relationships used for reasoning about observed object behaviour.

- The control and planning methodology concerns the task and strategic layers of HIVIS, describing how inference layer functions can be applied to fulfill some system purpose.

The interdependencies between the three methodologies are caused by the similarity of categorical coverage of spatial representation and events and behaviour.

Figure 2.1 illustrates the relationship between the two approaches to reasoning called "bottom-up" and "top-down", and the four layer model. As the figure illustrates, bottom-up reasoning is data-driven, while top-down reasoning is goal-driven. This four layer model corresponds to an operational ordering in the following way: The interaction between the spatial representation and events and behaviour methodologies takes the form of data exchange since both have domain and inference layer knowledge and, as such, are not
really affected by a switch between top-down and bottom-up reasoning. This is not the case when we consider how control and planning interacts with either or both of the other two methodologies. We have already had an example concerning control issues when we contrasted the open- and closed-loop architectures in figures 1.1 and 1.2. The feedback present in the closed-loop provided more opportunity for control to be exerted, allowing lower-level visual processing to be configured to a high-level task. In the surveillance problem we have two general options concerning the amount of control used. We can use the fixed task of identifying all behaviours, which would limit the complexity of control, or we could increase the complexity of control to identify fewer relevant behaviours, by selecting from a range of surveillance tasks. In the data-directed case control and planning have little effect, with the actions of the system determined by the supplied data. In the goal-directed case control and planning affect system performance by focusing upon data that is relevant to the goal.

In the following sections we will discuss approaches taken in previous work. These three surveys motivate the work done in the development of the computational approach and the implementation described in subsequent chapters.
Chapter 2. Knowledge in image understanding

2 Spatial representation

In this survey we first consider how to represent space in isolation of temporal considerations and then in section 3 return to consider these temporal aspects. This initial discussion of spatial representation seeks to identify one that fulfills assumption 3 (the static ground plane) where properties of the background environment remain the same over the duration of interest.

2.1 Surveillance

The discussion here is guided by requirements that enable a surveillance system to perform spatial reasoning about the moving objects that travel through the field of view. The spatial representation and reasoning needs to be expressive enough so that we can describe:

- The static environment that is visible to the perceiver.
- Single objects in terms of each moving object's spatial occupancy at a given moment relative to the environment. We wish to express the perceiver's knowledge of each object's position, its spatial extent, and the part of the environment that it occupies.
- Multiple objects in terms of their spatial arrangement at a given moment with respect to the environment and each other.
- The meaning of the different parts of the environment, including both physical properties and semantic properties. These semantic properties, although not physically present, are understood by the inhabitants of the environment and can be used by the perceiver to interpret observed behaviour.

In the realm of spatial representation these are unusual requirements and, as shown in figure 1.4, there is the option of operating in either (1) the image-plane of the perceiver (i.e., what the perceiver directly observes) or (2) a ground-plane projection which provides an overhead view that is not directly available to the perceiver. With these requirements in mind let us consider related work and identify which requirements they fulfill in our surveillance problem.

2.2 Computational approaches

When we consider that spatial forms are encountered in our everyday lives, it may not be surprising (as Sloman discusses in [217]) that there is such a diverse range of spatial representations, each created to support some particular purpose and viewpoint. These spatial forms seem at one and the same time unified by a common mathematical framework and yet separated by their everyday use. We can distinguish:

- The local space that is near us and in our field-of-view (visible-proximal space), e.g., the books, desk-top and coffee mug that are in reach and visible.
- The local space that is not currently in the field-of-view (occluded proximal space and out-of-shot proximal space), e.g., the under side of the table, the chair, the wall behind.
Chapter 2. Knowledge in Image Understanding

- The not so local space that is in our field-of-view but which we can not reach (visible space), this space can be very large (the view from a mountain top) but in our everyday experience is typically smaller due to occluding objects and quality of visual resolution.

- The not so local space that we cannot see (medium scale space) of the building we are occupying, the corridors and other rooms we can not see but know exist.

- The large scale space that connects the buildings, places and locations, the routes to work, home, the shops or another town that we have visited or know about. This distinction is the difference between the current location and memory of other places. See Lynch [148] for a description of how city dwellers remember their environment.

Surveillance is situated in the visible space of the perceiver and we are not usually concerned with medium scale space or large scale space. The static position of the perceiver (due to fixed camera), in this case, allows us to simplify our spatial representation.

When we turn to consider how to represent our local spatial form in a computer, using some formal language, we find that there are constraints caused by computer storage and execution speed, which place an upper bound on what is possible. The model of the “real world” held in the machine is not complete, being just a representation of those properties that are deemed necessary for the surveillance problem. The representation is biased towards understanding what takes place in the scene rather than providing a realistic display on the computer screen, and can be separated into (1) the static environment and (2) the dynamic objects that travel through the environment. The dynamic object descriptions, in the form of pose-positions, are the results of early and intermediate computer vision processes, and are intended to be independent of the mechanism used.

Here we review some related spatial representation schemes under the headings: large scale space; map learning; robotics and motion planning; graphics and solid modelling; qualitative reasoning about motion and arrangements; linguistic and cognitive approaches; spatial decomposition; intermediate vision. In the discussion below, we will consider only a small portion of the currently available literature. For more general reviews on spatial representation in the field of AI see Chen [38] and Davis [52].

2.2.1 Large Scale Space

The requirements of the surveillance problem generally exclude work on large scale space such as Geographic Information Systems (GIS) and cognitive maps. Laurini and Thompson [137] describe how GIS is more concerned with the representation of statistical and cartographic data. This data, particularly information about man-made structures (roads, etc.) may be of use when setting up the surveillance system but are unlikely to capture all the environmental details and may not correspond exactly to the “real world”.

Large scale cognitive maps (as typified by the collection of papers edited by Downs and Stea [60], and the computational models of Yeap [261] and Kuipers [130]) are more concerned with how we remember locations, including approaches like landmark identification (Lynch [148], Kuipers and Levitt [131]) and fuzzy boundaries (Davis [49]). These are representations of space that are intended to capture the uncertainty and distortions of
memory and perception. While cognitive maps may again be useful for setting up a surveillance system, the restricted field-of-view generally makes such larger frames-of-reference unnecessary during the surveillance operation. These approaches then are not needed for the surveillance problem, because we assume that the static environment is known to the perceiver (see assumption 3 given earlier).

2.2.2 Map learning

The problem of acquiring the data in the cognitive map is often related to the subject of large scale space. This concerns integrating the observed environment into a continuous memory structure (Davis [49], Yeap [261] and Mataric [153]). An example of such cognitive mapping, given in Hutchins’ paper [110] on Micronesian navigation, contrasts the Micronesian approach with Western map use (see for example Monmonier [162]). The Micronesian navigators use an egocentric perspective on the oceanic environment through which they are navigating. Compared with the Western use of map and compass the Micronesian navigators place a very different interpretation on the scene primarily based on ocean swells, the stars and the positions of reference islands.

An alternative to learning maps by exploration is learning maps from the observation of a fixed perceiver. Neumann’s group (e.g., [161]) have addressed this problem in the road traffic domain by learning the area of space through which vehicles typically travel. To do this, they use a spatio-temporal buffer, consisting of a matrix that accumulates velocities and orientation in each coordinate address, with each address being able to store multiple paths. The matrix does not represent time explicitly (making multiple vehicle reasoning impossible) nor is a vehicle’s extent represented. Although not suitable for the detailed representation of, and reasoning about, observed scene objects, this does support the formation of path predictions.

In the surveillance problem we assume that the map data is given a priori, so we do not address the map learning problem. It is included in the future work section. Learning the typical behaviour information via observation would address the complex and time consuming process of specifying the necessary data by hand.

2.2.3 Robotics and motion planning

Spatial representation in robotics (Schwartz and Yap [213]) and motion planning (Latombe [135]), use geometric concepts (Kapur and Mundy [115]). The problems addressed tend to be concerned with control issues, such as finding paths in the presence of obstacles for a movable object or manipulator. However, the surveillance problem does not really make use of path planning, although we are observing the activities of actors who are dynamically planning their path through their environment. Computing the feasibility of each object continuing on its path dynamically seems impractical, both in terms of the observer having access to the intentions of each object and the dynamic nature of the environment.
One approach that is useful for full path planning is called “configuration space” (Lozano-Pérez [147]). Obstacles in the environment of a moving robot are grown to reflect the extent of a moving robot and processing is then simplified to point motion. However, the configuration space approach is not well suited to the surveillance problem because of the existence of various object types, each of which would require a configuration space (for each degree of rotation). We are also reasoning about interactions between objects with no single central object of interest. We want to reason about all objects of interest equally, something which is difficult to do when using the configuration space approach, for example see Latombe’s description of multiple robots [135, pages 373–384].

Two approaches related to path planning which are worth noting, although not directly useful here, are nonholonomic models and retracts. Nonholonomic models (Laumond [136], Latombe’s [135, pages 403–451]) provide a more realistic description of paths formed by objects, such as cars, that have constants upon possible velocity directions (differential motions). Nonholonomic models provide an approach to generating better path predictions and checking the consistency of observed object motion.

Retracts (see Canny [32], Latombe [135], Spanier [218]) provide a simplified model of space by using a function, \( r : X \rightarrow A \) from a topological space \( X \) to a subspace \( A \subset X \), where \( r \) is continuous and is the identity on \( A \). When \( r \) exists \( A \) is called a “retract” of \( X \), and the map \( r \) is called a “retraction” of \( X \) to \( A \). Canny uses retraction to map 2D space onto a 1D subset, called the “roadmap”. For example, in the road-traffic domain, we could map the 2D space occupied by a road to a 1D line and describe the progress of the traffic on the road by points in the line. This would be useful for reasoning that needed to retain some features of the 2D space, e.g., the direction and position of the traffic on the 1D line, but which does not require reasoning about the local spatial arrangements of the objects.

Unlike the work on robotics described here the surveillance problem is not concerned with controlling the objects it is observing. Thus, although the representations discussed here provide possible approaches to predicting what the observed objects might do next, the computational costs involved make these approaches unattractive.

2.2.4 Graphics and solid modelling

The primary objective of the surveillance system is not to display a graphical representation of the world, so most issues associated with computer graphics are not applicable here. However, there is a common concern with how to represent and reason about geometric information. The books of Preparata and Shamos [189], Mehlhorn [157], describe geometric algorithms. Hoffmann [102] describes issues related to constructive solid geometry, Kapur and Mundy [115] provide links to work on vision and robotics. These geometric approaches do not themselves help solve the surveillance problem, but they can be useful as techniques to be applied in implementing the chosen spatial representation.
2.2.5 Qualitative reasoning about motion and arrangements

The collection of papers edited by Weld and de Kleer [248] contains a useful section on reasoning about shape and space. The papers cover various approaches to simulating a scenario for a given situation and saying what the outcome might be. The basic technique used typically employs "environment", which usually involves generating all the qualitatively distinct behaviours of a system for each possible initial state.

De Kleer’s NEWTON program [53] is an example of a topological spatial representation that uses environment. In NEWTON a rollercoaster ride is represented by an ordered sequence of regions such that, in each region, the values of "sign of curvature" and "tangent direction" are continuous (i.e., these two functions are used to define the extent of each region). The dynamic spatio-temporal aspect is described by "pre-compiling" all possible transitions between the regions into an environment which completely describes the world. However, environment is not good at handling very unexpected cases or interrelations between multiple objects in complex domains. Thus environment would only provide a useful model for describing the individual paths of well behaved actors. Unfortunately, in the everyday world, people often perform unexpected actions and it is precisely these actions that we want to identify. This makes environment, on its own, inappropriate for reasoning about everyday activity, although it may prove useful for reasoning about the relationships between event primitives (which are qualitative states).

Environment has been used to model more complicated domains, for example a 2D world in which the possible activity of a bouncing ball can be modelled (Forbus, Nielsen and Faltings [72]) and the CLOCK project [73]. A more formal definition of environment is provided by the axiomatic theory of Randell and Cohn [193, 194] who use a a first order predicate logic formalism. Their theory describes a qualitative ordering (held in a lattice structure) that connects topological changes in pairwise relationships to enable allowable transitions to be determined. There appears to be a substantial amount of theorem proving involved in this approach, which does not make it practical for a "real-time" problem like surveillance.

The CLOCK project is more concerned with kinematic analysis, other examples are given by Gelsey [83] and Kramer [128], which is a form of spatial reasoning with requirements quite different to those of the surveillance problem. Kinematic analysis is not appropriate to the surveillance problem because the observed objects are not part of a closed kinematic chain in a mechanism.

Freska [75] discusses some of the existing approaches to qualitative spatial reasoning and introduces the notion of qualitative orientation using an iconic representation that can be composed. For example, if we know the orientation of c given vector ab and we know the orientation of d given bc, Freska shows how to determine d with respect to ab. This might be useful for navigation, but is not needed in the surveillance problem.

These various approaches to qualitative reasoning can help with aspects of object interactions of the dynamic scene objects in the surveillance problem. They are not useful, however, for providing a model of the environment.
2.2.6 Linguistic and Cognitive Approaches

At a higher-level of reasoning we can consider how language is used to describe geographic space (Herkovits [98], Talmey [225]). In spatial representation and reasoning linguistic concerns are sometimes seen as more important than the geometric and topological relations that they are trying to describe. It is easy to ignore the fact that spatial representation and reasoning can be performed without language (Schöne [211]) and that most of the linguistic generalisations are to allow communication by using common-sense and commonly understood terms of reference that are made clear by their context. Work like (Lang et al. [133], Miller and Johnson-Laird [159]) provide a linguistic approach for describing geographic space which may be useful in communicating results and developing the terms from which conceptual descriptions can be formed. However, it is again the case that these approaches are not so useful for representing spatial data itself.

Piaget and Inhelder [186] provide a study on how children learn spatial concepts, using stages progressing from topology to Euclidean properties, corresponding to increasing age. This provides cognitive support for the use of topological reasoning in problem solving, which precede Euclidean relations in their simplicity. We will investigate the use of topological representations to determine if they offer any advantages in section 2.3.

2.2.7 Spatial Decomposition

Spatial data can often most easily be expressed in terms of a hierarchy based on enclosure. Rooms in a house, houses in a street, streets in a town, etc.. This has a obvious application in Geographic Information Systems (Laurini and Thompson [137]), and has also been used by Davis [49] as part of his map learning program MERCATOR. Thibadeau [232] describes a room decomposition (see figure 2.8) of an example used by Heider and Simmel which is described in section 3.2.2. This decomposition reflects the faces of the room rather than the typical paths used by the participants.

Spatial decomposition is a useful approach that is independent of the spatial primitives used, allowing semantic properties to be attached to spatial elements (such as rooms, houses, etc. in the example above). However, it only addresses a part of the problem that we need to address.

2.2.8 Intermediate-Level Vision

The representations of space used in vision research are not often concerned with modelling geographic or natural features, however, there are exceptions such as Pentland's [184] description of natural forms and the description of human motion by Badler and Smoliar [11], O'Rourke and Badler [179]). The majority of representations are concerned with supporting low-level vision (for example, Marr [150], Horn [103]). These approaches use geometric and topological spatial representations to describe the image features in the image-plane. One notable example is provided by Fleck [68] with her topological approach to representing digitised spaces for both edge detection and stereo matching. In general there is no commonality between the use of the spatial representation in low-level vision and
Figure 2.2: Interactive object modelling. Picture (a) shows the work window, and (b) shows fitted models. From Corrall and Hill [46].

high-level vision. However, as we describe later, there is commonality in the representation used.

Intermediate-level vision has much more in common with high-level vision, mainly because of the communication of results from intermediate-level visual processing. An instance of this is shown by how both are affected by assumption 3, which was made to enable model matching to be performed from single camera data. However, they concentrate on different properties of the scene. As shown in figure 2.2, intermediate-level vision can use a simplified 3D model of the static environment which includes the ground-plane over which the observed objects move and objects (such as the posts shown in figure 2.2(b)) that occlude some moving scene objects. Figure 2.2 shows the various 3D models and the results of fitting these to scene objects facilitating their recognition. The 3D models used here may use the the solid modelling techniques described in section 2.2.4. For further details see Murray and Buxton [169], Nagel [171], Koller et al. [124], and Worrall et al. [259].

One of the most wide ranging models of space is given by Fleck [68] in her topological approach to representing digitised spaces for both edge detection and stereo matching. This also contains applications of the topological approach to natural language and qualitative physics and is a natural candidate for extension to the surveillance problem.

2.2.9 Discussion

We have considered spatial representations both in terms of how they are viewed in the everyday world and how they have been viewed in a broad range of artificial intelligence
applications. We take note of the priority that Piaget and Inhelder [186] place on topological reasoning, and will take a closer look at Fleck’s [68] model of space because it supports both topological and Euclidean reasoning.

The scope and various uses of spatial representation outlined above have covered a broad range of purposes. However, not all of the requirements of the surveillance problem have been addressed. This is mainly because “purpose” is important in computational systems both natural and artificial. Excess functionality is typically present to tailor the representation to some particular spatial reasoning objective. Common to all approaches has been the use of a spatial representation although no single set of spatial primitives is evident. In the next subsection we propose a common representation for spatial representation that is suitable for the requirements of the surveillance.

2.3 A COMMON REPRESENTATION

In appendix B section 1, we develop a mathematical foundation that uses a topological approach which draws upon the work of Whitehead [251] and Fleck [68]. This mathematical foundation combines both topological and Euclidean properties so that we can make use of topological relations like closure, interior, boundary, separated and connected in conjunction with Euclidean distance. The basic element we use is called the n-cell where the n denotes the dimension of the cell. A 0-cell is a point or vertex, a 1-cell is a line, an edge or face of a higher dimensional object, a 2-cell is a surface or plane, such as the area bounded by a set of connected 1-cells which, as shown in figure 2.3 are themselves connected by 0-cells.

The representative power of this cellular approach is that it can be used to provide an abstract model that can be implemented in a variety of different ways. As such it can provide a common form to underly most spatial representations outlined above. Informally,
a cell is a blob of space, having some fiducial size. The cells do not need to have the same size but they will all be of some limited range of sizes reflecting the granularity of the structure of space. The basic idea is that we will build our spatial representation out of these cells. In the surveillance problem we have simplified the representation of space to 2D and use this to model the ground-plane and scene objects. Although reference is sometimes made to 3D forms they are not fully addressed and are outside the scope of this dissertation.

This common representation supports planar models of space and, as shown by Fleck [68], is suitable for representing problems from vision, qualitative physics and natural language. Figure 2.4 shows how cells can be used to provide a structure for space. This structure does not itself define any objects but is, instead used as a spatial framework to which we can attach information. In definition B.13 we introduce the term regular cell complex to describe a collection of disjoint cells such as those shown in the figure.

In chapter 3 on spatial representation we extend this model of space so that it can represent the various physical and semantic properties that are present in the environment under surveillance. These properties denote environmental features like walls and typical paths (for example, as shown in figure 2.5). The cellular representation can also be used to model the spatial extent of the dynamic objects (i.e., pose positions). In spatial terms this is not a complex issue. The difficulty arises when we consider its combination with time and the spatial arrangements of multiple dynamic objects. In the next subsection we will consider how this cellular representation can be implemented so that we can describe these environmental properties.

2.4 POLYHEDRAL TESSELLATIONS AND IMPLEMENTATIONS

In the previous subsection we introduced a cellular representation that uses a regular cell complex and which we now consider from the standpoint of implementation. In the surveillance problem we have both metric and topological data and will be using metric
and topological reasoning with this data to determine things like: the distances between
objects, and identifying the cells that each object is partially or completely occupying.

The cellular representation provides a topological foundation that directly supports
the reasoning required. Implementing this topological foundation needs to address issues
of efficiency by eliminating unnecessarily repeated calculations. These issues arise with
boundary representations (e.g., Davis [49], Hoffmann [102]) where topological relationships
need to be determined anew for each test via explicit calculation. It makes more sense to
use a representation that more directly describes the topological information.

In addition to these topological properties we also need to support metric data to allow
for Euclidean measures of distance and angles. The surveillance problem does not require
a perfect model of the real world, so we will approximate curves by using polyhedra, which
are much less complex in terms of implementation. We can compose a polyhedron from a
set of 1-cells, the connecting 0-cells and the 2-cells that the boundary 1-cell faces enclose.

**Definition 2.1** A *n* sided (simple) **polyhedron**, $\mathcal{P}$, is composed from a set of *n* 1-cell
faces, $\mathcal{L}_i$, such that $\mathcal{P} = \sum_{i=1}^{n} \mathcal{L}_i$, and that no members of the set $\{\mathcal{L}_1 \ldots \mathcal{L}_n\}$ cross (intersect).

This definition is important because it provides a geometric framework within which to
model the 2D ground-plane and also because it acts as a suitable interface to any space
structured using a regular cell complex. The definition does not exclude polyhedra with
concave vertices, and we note that such polyhedra tend to add complexity to implementa-
tion that support topological tests for occupancy. The implementation should not affect
the geometric map specification given in terms of polyhedra. The translation process from
a specification in terms of polyhedra to one in terms of regular cell complex can take one of
two approaches that are distinguished by how they structure the underlying space. These
approaches are called "regular" and "irregular".

### 2.4.1 Regular

The regular structure of space, sometimes called the "sugar-cube" model, represents 3D
space using a composition of regular volumetric shapes such as cubes (Koenderink [123,
page 48]) or tetrahedrons (Jung and Lee [113]). We can apply the same ideas to tile the
2D plane, using regular 2D shapes such as squares (or pixels in a "raster") and hexagons
(for a pixel level example see Bell et al. [19]), and an example is given in figure 2.4. The
main problem with this approach is that the structure of space does not always match the
object boundary that is being represented and results in "jaggyness". This approach to
structuring space is very attractive because of its simplicity, directly describing the spatial
extent of an area in terms of occupied cells, which gives this approach the name "occupancy
grid". It also marries well with implementations in terms of bitmaps, 2D arrays or hash-
tables of used addresses. When the granularity is too large, severe distortion and loss of
detail can result. The solution is to use a finer granularity, however, this can cause a major
storage problem. This storage problem can be greatly reduced by using quadtrees (Samet
(207)] or tesseral arithmetic (Bell et al. [18], Gargantini [82]). Both techniques can provide savings on storage of silhouette like shapes, where representation of an area is important. All members of this family of occupancy grids support the efficient implementation of set operations.

### 2.4.2 Irregular

Irregular tessellations can be used to provide a regular tessellation (i.e., they are the superset of all tessellation methods) and, if used in this way, would exhibit the same representational problems outlined above. However, irregular tessellations can also be used to provide a structure of space that complies with the spatial form of the environmental properties. This means that edges in the environment are represented by cell faces in the structure used to describe space. The presence of a matching structure removes the problem of cell granularity, as long as all the required cells are present in the supporting structure of space. There are two situations in which tessellations arise:

1. Given a set of points find a tessellation.
2. Given a set of non-intersecting polyhedra find a finer tessellation.

The spatial representations used in the surveillance problem are of this second kind where we use the known environmental structure (described in terms of polyhedra) to influence how the underlying cellular representation is to be tessellated. Many geometric algorithms perform better on convex point sets. Thus, it is expedient to decompose concave polyhedra into combinations of convex polyhedra. The most primitive is the triangle, for which there are a number of triangulation algorithms, some of which are discussed in appendix B section 2.4.

There are other approaches to decomposing space, such as exact cell decomposition which requires as input a set of algebraic equations (which we can check for consistency by using a Gröbner bases algorithm (Hoffman [102, pages 257-300], Bruchberger [28])), and for which the method partitions the space outside each defined polyhedra into a finite collection of semi-algebraic cells (the complement of polyhedral decomposition). For example in figure 2.6, f is the equation \( x^2 + y^2 - 1 = 0 \) (the unit disc) which, as shown in figure 2.7, can be recursively decomposed into thirteen cells (two 0-cells, six 1-cells, five 2-cells) by using Collins' decomposition algorithm (Collins [44], Latombe [135, pages 225–242]) that employs a method based on Tarski's [227] approach for deciding the satisfiability of Tarski sentences.

Once we have our cellular decomposition, we then have the problem of how to store the points, lines, and areas in various data structures. We could adopt a winged edge model such as the one Paoluzzi et al. [181] use in their implementation of simplicial complexes. Another promising approach is described by Günther [92, 93], who uses the properties given in definition B.8 to combine edges in the form of hyperplanes so that they model the polyhedra without the need for specifying vertices. Also, since we are talking about combinations of cells, we could treat the cells as constructive solid geometry (CSG) primitives (Hoffmann [102]) so that a combination of cells would be represented by
a binary tree with union operations at the nodes. These three examples provide an insight into the range of options available.

2.5 Summary

In this review of we have seen that spatial representation is a common theme running through a number of areas in AI and computer science. Here spatial representation has mostly concerned data held at the domain-layer. We will consider issues relating to dynamics in section 3. In this section we have already begun to make some selections. For example, the adoption of the cellular representation, the use of polyhedra to specify the geometry of the 2D ground-plane, and the use of a mapping from the polyhedral specification to a cellular representation that is to be used by the runtime spatial reasoning system. The choices that remain concern how the cellular representation is to be implemented, the development of a database to hold the spatial information about the static environment, and the provision of an interface between the database and the rest of the runtime system allowing communication about the dynamic object data. All the regular and irregular implementations would work since they support the cellular representation we introduced above. Some approaches are more attractive than others in terms of storage, runtime performance or additional representational power. In chapter 3 section 3 we discuss two such implementations.
3 EVENTS AND BEHAVIOUR

The discussion of spatial representation ignored issues associated with the full representation of time and reasoning about dynamic properties. In this section we will address these. The objective of this review section is to describe approaches that are concerned with the representation and reasoning of events and behaviour (i.e., the domain- and inference-layers). We will more fully address related control issues, such as the mechanisms that operate on these representations in section 4.

3.1 SURVEILLANCE

In the context of surveillance we encounter a problem common to any system that is situated in a dynamic environment. This is how to represent and reason about properties in the environment that change over time. There is a whole area of AI and logic devoted to representing and reasoning about time. However, when we consider the issues from the surveillance standpoint, the requirements of these complete temporal reasoning mechanisms seem overly complex. While fine for reasoning about the situation in hindsight and planning what to do next, a less complex notion of time and events better captures the dynamic reasoning encountered here. Some researchers have taken this approach a stage further and considered whether representing (modelling) the external world is necessary. In robotics, letting the world “be its own model” (Brooks [24]) can simplify their problem. However, in surveillance we need to represent the properties of the the dynamic environment that are potentially relevant to the surveillance task. Remembering these properties is to enable us to identify when a change in some property occurs. The context in which the change occurs is also important, since it is the context that provides meaning to the change. In the surveillance problem part of the context is provided by the underlying static environment.

In this section we are considering previous work on representing and reasoning about events and behaviour. We will first of all consider what needs to be represented under the headings of: (1) events, episodes and verbs; (2) perception of events; (3) the deictic viewpoint; (4) situatedness. Once this background has been covered, we will also consider options for extending our spatial ontology to include time with the objective of providing an integrated spatio-temporal representation of the motion of each dynamic scene object (or agent) in its everyday activity. This discussion takes place under the headings: (1) temporal logic; (2) tracking.

3.2 COMPUTATIONAL APPROACHES

3.2.1 EVENTS, EPISODES AND VERBS

The term “event” is widely used yet has no specific definition. This categorisation, called “event”, provides a useful denotation for describing everyday happenings, allowing the continuity of everyday experience to be cut up into discrete bounded temporal units. An event is often used to denote a unit of action which can be placed in a predictable order.
Schank and Abelson [209] compose events into scripts to describe typical behaviour of a customer at a restaurant such as entering, going to a table, ordering, eating and paying.

To describe the behaviour of agents we are using an ontology based upon that described by Nagel [171], which captures the common sense notions of the terms being used. The data supplied by the intermediate-level vision component is considered to be a signal from a sensor however complicated that sensor may be. The first thing we look for in this signal are changes that differ significantly from background noise, where noise is defined as some known property associated with the sensor. A signal change is the most primitive element used by this system. An event is any change that has been given a significant predefined semantic, providing the particular change with a symbolic record in the system.\(^1\) To describe combinations of more than one event we will use the term episode to cover both simple and complex sequences of action. The reason for this denotation is to make events our primitive unit of change associated with the visual system. This provides a hierarchical layering from events to episodes, and then to scripts. This hierarchical decomposition and relationships between the behavioural elements can be used to define a grammar where events are terminal symbols in the language to be parsed. This approach could use the syntactic methods described by Fu [79], the static semantics of an attributed grammar as described by Frost [78], the island parsing described by Corrall and Hill [46], or the compositional semantics described by Woods [258], Pereira and Warren [185] and Dowty, Wall and Peters [61]. Each of these supports a bottom-up model to form conceptual descriptions.

Badler [10] is one of the first researchers to address the problems associated with extracting a natural language description that captures the activity of the moving objects present in a sequence of images. For descriptive purposes, he used an animation of a natural scene that provides metrical spatial input. Badler provides a general purpose representation of motion and spatial relationships based upon motion verbs, and the representation used combines the metrical input with a semantic network that could then be used as input to a text generating algorithm. There are problems with Badler’s approach because of the accuracy required from the proposed vision system which is used to supply the necessary metrical data to his algorithms. This makes the algorithms impractical as vision is uncertain and incomplete. However, the account given of how motions can be aggregated into more abstract, meaningful ones is valuable.

Badler’s work has been developed further and implemented in the ALVEN system (Tsotsos [237, 239, 241]) which uses a realistic vision system with provision for noise and occlusion. Its final output is a frame representation from which text could be generated. The motion verbs used by Badler and Tsotsos originated in work by Miller, which was later extended in joint work with Johnson-Laird [159]. This formal analysis of motion verbs uses a logico-linguistic approach together with an informal computational framework. This

---

\(^1\) In past work (e.g., [105]) I have often used the term event to mean both “instantaneous” change and combinations of events. From now on an event will just refer to a signal change of semantic relevance.
has provided the foundation for other work that also draws on the linguistic literature to ground prepositions in the spatial primitives described by Herković [98]. Two such projects are described by Nagel [171] (NAOS by Neumann[173] and CITYTOUR by Retz-Schmidt [198]) which are related by their common use of the road traffic domain. Both NAOS and CITYTOUR are concerned with providing natural-language descriptions from sequences of images of moving objects, that allow question-answering to take place as an off-line user query process.

Describing events by using verbs is useful because we do this in our everyday lives, however, the flexibility of language may not be needed to describe perceptual events which are really pre-linguistic. Some of the problems of specifying events are described by Renolds [199], who recounts issues found when combining categorisation schemes developed by different observers to describe the behaviour of Rhesus Monkeys. The selection of verbs depends upon the task of the observer. The decomposition used to convert observations into events and episodes depends on what is considered to be a useful conceptual unit. For example, in some situations the episode “eating” might be a useful conceptual unit, but could itself be decomposed into finer episodes that could be individually perceived should the perceiver have both the opportunity and wish to do so.

Neumann [173] has developed the list of motion verbs shown in table 2.1 to provide natural language descriptions of activity in the road traffic domain. The only way to say that this list is complete is to assume a closed world because as shown in table 2.2, there are other verbs that might be more applicable (these are from a list developed by Badler). Also the allocation of a correct verb is difficult because different verbs may be appropriate depending on the length of time a particular activity takes.²

We will call these the “script-based” approaches because of their goal of natural language generation.

### 3.2.2 Perception of events

Experimental psychology has been used to address the problem of how events are perceived. Bruce and Green [26, pages 311–374] provide an overview and, here we will pick out a few notable examples that have investigated how causal relationships are perceived. The first is based on Michotte’s launching experiments [158], and the second on Heider and Simmel’s work on apparent behaviour [96]. Michotte has investigated the effect of varying the temporal interval between “action” and “reaction”. These launching experiments employ the movements of simulated blocks, which bring to light properties of perceptual control (further described by Weir [247] and Leslie [141, 142]) but are not that illustrative in terms of everyday events. A more appropriate example is provided by Thibadeau. Thibadeau [232] describes a computational tool that provides a conceptual description of Heider and Simmel’s film in which three “agents”, represented by a large triangle, a small triangle and a

²Another illustration of correct verb allocation occurs when linguists describe one language in terms of another, because languages do not always have a one-to-one mapping. For example, the number of words in Eskimo for the word snow (see Thalbitzer [231] and Simpson [215, page 188]).
<table>
<thead>
<tr>
<th>node</th>
<th>parents</th>
<th>children</th>
</tr>
</thead>
<tbody>
<tr>
<td>accelerate</td>
<td>drive</td>
<td>depart, start-driving</td>
</tr>
<tr>
<td>approach</td>
<td>move</td>
<td>meet, catch-up-with, reach, pass, drive-past, go-past</td>
</tr>
<tr>
<td>arrive</td>
<td>come-near</td>
<td>come-near</td>
</tr>
<tr>
<td>catch-up-width</td>
<td>approaches</td>
<td>come-near</td>
</tr>
<tr>
<td>come</td>
<td>move</td>
<td>arrive</td>
</tr>
<tr>
<td>come-near</td>
<td>come</td>
<td></td>
</tr>
<tr>
<td>continue-driving</td>
<td>drive</td>
<td></td>
</tr>
<tr>
<td>continue-walking</td>
<td>walk</td>
<td>accelerate, slow-down, drive-behind, follow, speed, drive-past, drive-off, continue-driving</td>
</tr>
<tr>
<td>cross</td>
<td>move, reach, leave</td>
<td></td>
</tr>
<tr>
<td>depart</td>
<td>accelerate, halt</td>
<td></td>
</tr>
<tr>
<td>drive</td>
<td>move</td>
<td></td>
</tr>
<tr>
<td>drive-off</td>
<td>recede, drive, halt</td>
<td>drive-round, overtake</td>
</tr>
<tr>
<td>drive-past</td>
<td>recede, drive, approach, pass</td>
<td>move, stand</td>
</tr>
<tr>
<td>drive-round</td>
<td>drive-past</td>
<td>drive-behind</td>
</tr>
<tr>
<td>exist</td>
<td></td>
<td></td>
</tr>
<tr>
<td>follow</td>
<td>move</td>
<td>walk-round</td>
</tr>
<tr>
<td>go-off</td>
<td>walk, stand</td>
<td>depart, start-driving, stop, park, drive-off, cross</td>
</tr>
<tr>
<td>go-past</td>
<td>recede, walk, approach, pass</td>
<td></td>
</tr>
<tr>
<td>halt</td>
<td>stand</td>
<td></td>
</tr>
<tr>
<td>leave</td>
<td>recede</td>
<td></td>
</tr>
<tr>
<td>make-u-turn</td>
<td>turn-round</td>
<td></td>
</tr>
<tr>
<td>meet</td>
<td>approach</td>
<td>yield/avoid, turn, recede, drive, follow, walk, come, run, approach, speed, stop, cross</td>
</tr>
<tr>
<td>move</td>
<td>exist</td>
<td></td>
</tr>
<tr>
<td>overtake</td>
<td>drive-past</td>
<td>drive-past, go-past</td>
</tr>
<tr>
<td>park</td>
<td>halt stop</td>
<td>cross</td>
</tr>
<tr>
<td>pass</td>
<td>approach, recede</td>
<td>pass, leave, drive-past, go-past, drive-off</td>
</tr>
<tr>
<td>reach</td>
<td>approach</td>
<td></td>
</tr>
<tr>
<td>recede</td>
<td>move</td>
<td></td>
</tr>
<tr>
<td>remain-standing</td>
<td>stand</td>
<td></td>
</tr>
<tr>
<td>resume-walking</td>
<td>start-walking, stand</td>
<td></td>
</tr>
<tr>
<td>return</td>
<td>turn-round</td>
<td></td>
</tr>
<tr>
<td>run</td>
<td>move</td>
<td>run-fast</td>
</tr>
<tr>
<td>run-fast</td>
<td>run</td>
<td>stop</td>
</tr>
<tr>
<td>slow-down</td>
<td>drive</td>
<td>halt, start-walking, stop, remain-standing, wait, go-off, resume-walking, resume-driving, resume-walking, park</td>
</tr>
<tr>
<td>speed</td>
<td>move, drive</td>
<td></td>
</tr>
<tr>
<td>stand</td>
<td>exist</td>
<td></td>
</tr>
<tr>
<td>start-driving</td>
<td>accelerate, halt</td>
<td></td>
</tr>
<tr>
<td>start-walking</td>
<td>walk, stand</td>
<td></td>
</tr>
<tr>
<td>stop</td>
<td>move, stand, slow-down, halt, stop</td>
<td></td>
</tr>
<tr>
<td>tread-on</td>
<td>walk</td>
<td></td>
</tr>
<tr>
<td>turn</td>
<td>move</td>
<td>turn-off, turn-into, turn-around</td>
</tr>
<tr>
<td>turn-into</td>
<td>turn</td>
<td>return, make-u-turn</td>
</tr>
<tr>
<td>turn-off</td>
<td>turn</td>
<td></td>
</tr>
<tr>
<td>turn-round</td>
<td>turn</td>
<td></td>
</tr>
<tr>
<td>wait</td>
<td>stand</td>
<td></td>
</tr>
<tr>
<td>walk</td>
<td>move</td>
<td></td>
</tr>
<tr>
<td>walk-round</td>
<td>go-past</td>
<td></td>
</tr>
<tr>
<td>yield/avoid</td>
<td>move</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Event hierarchy in NAOS. From Neumann [173].
absorbs empties launches pulls shakes stumbles undulates
accompanies enters leads pursues shifts substitutes vacates
admits escapes lengthens pushes shoves swerves veers
advances exits limps puts shrinks swims visits
ambles expands lumbers races shrugs takes voyages
assembles extends lurches rambles skates throws wades
attracts fills marches releases slackens thrusts walks
broadens flaps meanders removes slides tilts wanders
canter flies nears replaces slithers tiptoes waves
canter flexes opens retreats spins toddlers weaves
closes gallops oscillates revolves spirals totters whirls
collides gathers paces rolls staggerers tours widens
continues goes parallels rotates stationary taws withdraws
crawls grows penetrates sails starts travels wobbles
dances injects pivots saunters steps traverses worms
decelerates inserts places scatters still streets wriggles
drifts interposes precedes scrambles stragglers trots zigzags
ejects jerks progresses scurries strays trundles
embarck jogs projects sends strides twirls
emits journeys propels separates strolls twists

Table 2.2: Example motion verbs.

disc, are situated in a 2D world as shown in figure 2.8, in which the rectangle with a moving flap is called a house. The movement of the various shapes are described in [96, 232]. In Heider and Simmel's experiments, even in the experiment where the subjects were not prompted, a large majority of the subjects perceived the activity of the agents in terms of animate beings, chiefly of persons, spontaneously attributing intentions to the moving parts of the artificial display. Thibadeau provides an account of how action perception can be performed without object and motion perception. Action perception involves discrete motion of objects, with emphasis placed on the conditions for the beginning and ending of motions. He uses Newton's "point-of-action-definition", which denotes a simple and direct report from a person about when the action is seen (for details see Newton [175]). A temporal structure is used where an action separates a temporal history into prior states and posterior states, with the point-of-action-definition occurring just after the event has happened. The results of Newton's work are two fold, first, that people normally segment behaviour into actions, even when they are not required to do so during an experiment, and they are remarkably unaware of their segmentations. Second, that expectation strongly affects action perception because viewers must be prepared to see an action in order for them to see it. We will return to this property of perceptual control in section 4.

Other experiments on understanding behaviour are described by Newton [175] and From [77] who have both analysed film sequences of people performing various everyday activity which has similarity to "ethology". The commonality is because they all study the behaviour of animals in their normal environment however, ethology also includes the problem of understanding non-humans (see Bekoff and Jamieson [17], Reynolds [199]).
Figure 2.8: (a) A frame from Heider and Simmel’s film [96, page 244] showing the three agents and the house. (b) From Thibadeau’s implementation [232, page 146] showing the spatial decomposition used in his analysis of the film.

To legitimise the perception of events, Gibson [85, pages 93-110] places it in the wider context of the environment, describing how an event seen by the perceiver corresponds to a disturbance in the invariant structure of the perceiver’s optic array. This provides a more complete theory for how events are perceived. Leyton [146] presents a different perspective on causality by saying that causal histories are implicit in the contours of the shapes or the arrangements of the shapes that we see. Knowing the causal history from the current observation would assist in the production of predictions and a deeper understanding of the current situation. One way of beginning to represent this causal knowledge would be to use an envisionment (Forbus [72]) of all possible causal relationships and from these to analyse what is happening. This provides a fixed model of all the possible causal histories, and has proved useful for analysing mechanisms and could be used to analyse the block experiments of Michotte. However, it is not suited to the dynamic nature of the surveillance problem. From a more philosophical standpoint Dretske [62] investigates the relationship between behaviour and causality.

These psychological experiments investigate causal properties in isolation and, while it would be wrong to say that these causal relationships do not exist, they do not need to play a prominent role in our implementation. Instead they used to motivate the ontology in which significant changes are called “events”. Thibadeau provides a useful framework for this ontology when he recognises two classes of change in a succession of states: motions, which are first-order change descriptions; and actions, which are second-order change descriptions. He develops a theory using only properties of second-order change so that the theory preserves direct perception of actions and does not use motion or object perception. In the context of attentional control, Birnbaum et al. [20] describe how the causal information present in a scene can be used to provide an ordering on how its contents are attended. They illustrate this using a scene containing block structures.

Here we have covered the perception of events and causality, which we will call the “causal” approaches.
Figure 2.9: The different coordinate systems of frame values used by the official-observer and the vehicles it is watching.

3.2.3 The deictic viewpoint

The surveillance system consists of a perceiver observing a scene which provides one frame of reference, that of the perceiver. However, there is also the frame of reference of the scene objects, particularly if they happen to be people (or people related objects). Figure 2.9 shows an example drawn from the road traffic domain. The perceiver-centered frame is called "deictic", the environmental-centered frame is called "extrinsic", and each object-centered frame is called "intrinsic" to the object concerned. In the figure the intrinsic frame of each vehicle coincides with the deictic frame of the driver of the vehicle (i.e., the front, sides and rear of the vehicle concerned).

Deixis is used in several disciplines such as anthropology (Hanks [94]), linguistics (Bühler [29]), the social sciences (Garfinkel [81], Heritage [97]) and spatial representation (Retz-Schmidt [198], Horkovits [98, pages 156–192]). Levinson’s book [144, pages 54–96] provides a useful introduction, providing an account of its use in natural language which is not covered here. Deixis is the use or referent of a deictic word (from the Greek word, deiknunai, meaning "to show", "pointing" or "locating"). A deictic word (e.g., I, now, this, that, here) is an aspect of a communication whose interpretation depends on knowledge of the context in which it occurs. Much of the research on deixis can be considered to comply with three orienting assumptions which we will call concreteness, subjectivity and functional isolation. The concreteness of deixis captures the sense of "here-and-now". We all have bodies that live in space, have perceptual fields, have motion and engage in various everyday activities. We can recall the past and anticipate the future, stretching time out from the "here-and-now" that we inhabit. These facts do not call into question the sense of immediacy that attaches to the "here-and-now".

Paired with the concreteness of body space is the natural assumption that the centre of space is the individual, providing a zero point like reference. Thus the space of "here" takes on a subjective appearance. The tendency to view immediately experienced space
from an individual angle imparts a subjectivity and egocentric quality into the notion of “here”. Deictic terms merely designate an object without describing it in any way. On each occasion of its use it applies to only one thing but may apply to different things on different occasions. This property is called functional isolation. This means that under Tarskian semantics, where symbols are considered to be rigid designators that always refer to the same object in the world, deictic terms are not symbols. Interestingly, Newell and Simon [174] use a different definition of symbol (which we will call the “NS-symbol”) that incorporates pre-linguistic as well as the usual meaning from logic and programming languages. NS-symbols can cause problems when used to describe deictic representations (for example, see Vera and Simon’s [245]). In general, the viewpoint provided by deictic representation is different from the global “state-based” view usually taken in AI because we are interpreting information in a local, body-oriented way.

In the description so far we have given some examples of spatial-deixis (e.g., “here”, “there”), temporal-deixis (e.g., “now”, “today”) and personal-deixis (e.g., “I”, “you”). Another point is that some objects have an “intrinsic-front” due to the concreteness assumption (e.g., a chair and a vehicle each have an intrinsic-front where an object such as a rock does not). When an object has an intrinsic-front we can allocate a set of “field values” (e.g., left, right, top, bottom, front, behind, near, far) providing a frame-of-reference for it. A human being learns to construct a frame-of-reference starting from two basic experiences: (1) the experience of looking straight ahead with his or her body standing upright on horizontal ground (we will call this the “canonical position”), and (2) the experience of encountering another human being face-to-face (the “canonical encounter”). In figure 2.10(a) we see the horizontal plane with the observer (denoted by an eye shape) looking forward. The horizontal axes are described by terms like left/right, front/back/side, before/behind. 3 In figure 2.10(b) the perceiver in effect “combines” the point of view of the person encountered with his or her own, thus the axes are taken about Mary. The front and back axes are Mary’s and point in directions opposite to those of the onlooker. However, the right and left axes have the same direction as the observer’s right and left, the opposite of Mary’s right and left. This symmetry makes left/right distinctions hard to learn and even adults can confuse the two. Hermits [98] says that the difficulty would probably be overwhelming if, besides drawing the distinction correctly on themselves, speakers and hearers had to reverse right and left, as in figure 2.10(c). However, “basic order” is more useful because it does not cause problems with perceived objects that change between facing and not facing the observer. Bühler [29] also mentions this basic order model when he describes a gymnastics teacher facing a dressed line of gymnasts and giving commands where the orders left and right are conventionally given and understood according to the gymnasts’ orientation. Bühler notes the astonishing ease of translating all field values of the visual system and the verbal deictic system from someone in another plane of orienta-

---

3In this dissertation we do not consider the vertical axis and will not use terms like top/bottom, above/below, over/under, even though this information might be available to the official-observer.
Bühler's description appears to be at odds with Herskovits. However, we can resolve this by referring to figure 2.9 and identifying that Herskovits is using the perceiver's frame of reference and that Bühler is using the intrinsic frames of reference. A difference between "global" and "local" reasoning which is used later in chapters 4 and 5.

By having access to the frame-of-reference of a perceived object, we can analyse all the changes involved or transpositions through space of such an object in terms of our own field values and experience. This provides a more natural feel because it is the system that we use in our everyday interactions with the world. Linguistic formalisms have shown that the use of deictic-forms is not straightforward. They do not easily fit into a theory that describes other referential forms that are not so dependent on context. Some approaches to this problem are described in Dowty, Wall and Peters [61, pages 136–139]. Subramanian and Woodfill [222] discuss one approach to converting deictic-forms to situation calculus [154] in which they provide implicit quantification, with the actual referent of the deictic-form determined by the perceptual machinery of the agent. One advantage of using a deictic representation is that it allows a propositional theory to be developed that is proportional to the number of properties of interest, as opposed to the number of propositional objects in the world. This means that in surveillance we do not need to provide a unique name for every propositional object that will ever pass through our field of view. Instead we can define a fixed, smaller number of properties of interest that the official-observer uses to describe the activities of the scene objects. We will call this use of deictic representation the "indexical" approach.

3.2.4 Situatedness

All the agents that the official-observer perceives are engaged in activity that is situated in the environment. Each agent has goals that are unknown to the official-observer, who
only sees the interactions between the agents and the environment. Winograd and Flores [256, pages 27–37] describe how the interpretation of the perceiver is not neutral, it is fundamentally social. The activities of the agents are not planned out in detail, instead they are in a state of “thrownness” (for details see [256] for a description of Heidegger’s philosophy). “thrownness” is Heidegger’s term for expressing the everyday experience of continual change, which is particularly apparent when interacting with other people (Winograd and Flores use chairing a meeting as an example). When interacting it is not possible to step back, reflect and plan. Suchman [223] investigates the issue of plans as something that can evolve out of “situated activity” and the use of previous experience to structure future activity. This can take the form of routines or more abstract plans. These abstract plans may have an organisation like scripts mentioned is section 3.2.1. In this form they provide a mechanism for thinking about actions, not performing the acts themselves. Using scripts or rules to govern behaviour would remove much of the dynamic improvisational quality of social activity. To more fully understand the activity of actors in the scene it seems appropriate to investigate approaches that take account of the improvisation possible in social activity. Although we cannot fully model social activity in this work, we have developed the more situated approach in chapter 5 based on the idea that the observed agents are indeed improvising their activity. We assume (see assumption 8) that we can use simple local models that reflect how the official-observer interprets this perceived activity.

3.3 Temporal Logic

In the surveillance problem, we do not require a full temporal reasoning ability although the reasoning that is performed clearly has an important temporal component. The actions that we are reasoning about are the “primary realities” and time is really an abstraction from them. The concept of time as a periodic process is drawn from the regular repeated ticking of clocks (see, for example, Reichenbach [195, pages 115-117]). The activity of the scene objects is perceived, but time is not. The intermediate-level vision component provides a sequence of frames marking out a non-reversible stream of spatio-temporal vehicle updates. This provides an unnecessarily discrete view of the world because the perception of physical motion is not inherently discrete. For example, if some scene object were to oscillate at a frequency some integer multiple of the frame rate then the oscillating object would appear stationary, even though it is not. The physical motions we perceive are characterised by cyclic or continuous functions that have no natural beginning or ending. For example, in the road-traffic domain, most of the vehicles travel in one continuous motion during the time that they are visible to the observer. From this observation of physical motion, as we discussed in section 3.2.2, the observed behaviour is segmented into actions, with each having a clear beginning and end (see Newton’s [175] for further details). Here we want to investigate previous temporal representations to see whether they support the discrete input data, continuous properties implicit in the data, and the denotation of actions.
Detailed discussion of temporal representation is outside the scope of this dissertation, see Allen [5], McDermott [155], Shoham [214] and van Benthem [244] for further details. Although full details will not be covered, we still need to consider how time can be represented. The above AI approaches all deal with the assertion of truth about a property over some measure of time. This can provide the foundation of temporal assertions, building an evolving database of temporal properties that could then be used for temporal reasoning. We consider the control issues raised by such a mechanism in section 4.

There are three approaches which we call “pointwise”, “interval-axiom” and “cellwise”. The pointwise approach is due to Taylor [229], the interval-axiom one covers most AI temporal representations and is based upon work by Allen [5], and the cellwise approach applies the cellular topology (introduced in the previous section) to the time-axis. We discuss these further in appendix C pointing out some of the problems with each approach and showing that the cellwise model of time is better, in some respects, for addressing the surveillance problem and that it also has the added benefit of integrating well with our spatial representation, providing a unified model of space and time. In the cellwise model we are able to represent each frame-update as a cell within a cell complex whose underlying space is the real number line $\mathbb{R}$. The additional cell structure is introduced to allow us to represent the beginning or ending of properties that are identified during spatio-temporal reasoning. Cellwise time provides a model of time that supports the representation of discrete frame updates and any continuous properties identified from these updates.

3.4 Tracking

We have now described how time can be represented, however, this does not address how we represent the spatio-temporal properties of the dynamic scene objects. We want to consider how to represent the continuous path that each object sweeps out as it travels through the field-of-view of the official-observer, and also the expectations that each object's motion generates. Some parts of this tracking process are best performed by the intermediate-level vision component that has access to the evolving results from the image data. Dealing with compact encodings from intermediate-level vision does not provide a good foundation for some of the approaches that we might consider using. This also means that we do not have to address the problem of motion correspondence (see Cox [47] for details), since this is mostly resolved before the compact encodings are generated. There is the possibility of using the knowledge present in the high-level component to resolve some occlusion problems, but we do not address this issue here. We can separate the tracking problem into three approaches called “spatio-temporal”, “valence”, and “markers”.

In the spatio-temporal approach the concern is modelling the path swept out by the object. In general this is a 3D+t or 4D problem as described by Dickmanns and Myshiwetz [57] however, here we make the simplifying assumption of operating in the ground-plane (see assumption 2). There are a number of ways of modelling the dynamic spatio-temporal data such as building a 2D+t representation from the pose-boxes, or using the object’s
centroids in a simplified 2D+t representation. To correct noise present in the data, maneuvering target tracking could be used. Bar-Shalom and Fortmann [15] describe a number of options like the Kalman filter, variable dimension filter and $\alpha\beta\gamma$-filter. The result of the target tracking could then form the basis for a 2D+t representation. Alternatively, the history represented in the state and state-transition matrices may be sufficient. Although useful for tracking object maneuvers, these filter mechanisms do not in themselves provide all the necessary spatio-temporal data necessary for reasoning about the activities of the scene objects. This spatio-temporal representation is considered further in chapter 4.

The valence approach is described by Gibson [85] where he discusses how spatio-temporal predictions might be formed. The idea behind this is that in some cases the outcome of an event sequence is implicit at the outset so that it is possible to foresee the development, and possibly the end, when a perceiver sees the beginning. This is not a static problem, instead it is continuously evolving. Figure 2.11 provides a road-traffic domain example of how vehicle paths are seen and how anticipated locomotion is modified by obstacles in advance of the actual process. Gibson describes how the totality of possible paths acts as a guide to locomotion, being the process that keeps you to the middle, as shown in the figure where the vehicle is kept in the centre of the field of safe travel. See Gibson [85, pages 223-237] for discussion on this and other optical information necessary for control of locomotion. Should we want to implement the vehicle path part of this valence approach we could use the maneuvering target tracking mechanisms to provide an estimate of the field-of-travel for an object even though these filters are poor at providing predictions (Goodwin and Sin [88] describe some better solutions). It is certainly the case that expectation plays a key role in understanding object activity in the surveillance problem. However we will not discuss all aspects of expectation, instead we briefly cover some parts of it in appendix C when we consider its relationship to map learning.
The marker approach does not retain any temporal history. It is closely associated with the attentional mechanisms used in early vision which have been investigated by Koch and Ullman [122] (which Chapman [34] has implemented) and Mozer [165] and Tsotsos [238] (which he has implemented with Culhane [48]). The attentional mechanism is used to tune the early visual input, selecting a small portion of the visual stimuli to process. The compact encodings provided in the surveillance problem could have been identified using an attentional approach, although the current problem description does not allow feedback for directing the early visual attentional processing. We can approximate this, using an approach described by Agre [2] and Chapman [34] where a buffer is used to store the new frame-update for each object under some form of unique identifier that allows its contents to be updated with each new frame. Here the unique identifier acts as the pointer address that can be held by a marker. No frame-update history is maintained and the changing object position values give the effect of the marker tracking the object. We discuss this approach further in chapter 5.

3.5 Summary

In this review of events and behaviour we have dealt with the representation and reasoning that occurs at the domain- and inference-layers, and in chapter 4 we will investigate the use of cellwise time, spatio-temporal paths, markers and the ontology of events and episodes. This investigation will address how units of action are identified from the results supplied by intermediate-level vision, and whether waiting until all required events have been observed is necessary for identifying an episode.

Of the four background approaches (script-based, causal, indexical and situated) only script-based and situated provide distinct frameworks that are incompatible with each other. This difference is due to their respective emphasis of off-line and on-line reasoning. The script-based approach provides a suitable framework for designing our first attempt at the surveillance problem because it uses conventional AI techniques. There are problems with this approach. We found that reasoning in retrospect, in effect off-line processing, does not fully address the surveillance problem. However, such an off-line explanation facility would be a useful capability in an operational system to assess how runtime reasoning performed. To make a script-based more on-line we will consider in section 4 (when we describe time maps and plan recognition) the use of partially instantiated scripts for predicting what is to happen next, and also to determine what is happening now.

Once we identify deictic representation as being an important component of surveillance, we find that this needs the on-line reasoning provided by the situated approach. The combined indexical and situated approaches present an appealing framework in which to investigate a more dynamic approach to surveillance because they are able to capture the improvised interactions present in everyday activity. The adoption of the indexical approach brings with it the requirement of more complex control. This design requires considering task- and strategic-layer knowledge; the topic of our next review section.
4 CONTROL AND PLANNING

The previous two sections have dealt with the domain- and inference-layers, and now we move our discussion to the task- and strategic-layers. These different categories of knowledge concern the task of understanding the data being processed. Task knowledge involves control and planning, in particular the role that interpretation plays. Task knowledge is important. For example, different tasks make different object properties apparent: we can use a cup to drink out of or to hold pens or to throw or to build a tower. These cases consider its "ability to hold a drinkable liquid", its "containership", its "throwableness", its "stackableness". If we give the official-observer a task we create prior and evolving expectations that cause some features to become important. We touched upon this in the discussion of Newton's work (section 3.2.2) where expectation was found to strongly affect action perception, and also in section 3.2.4 when we described situatedness.

These issues about task knowledge are relevant to the surveillance problem because, although we have a static camera and no effectors that can act upon the environment, we can select which objects to attend within the field of view.

4.1 SURVEILLANCE

In section 3 we identified script-based and situated approaches for off-line and on-line reasoning. By putting the interpretation off-line we remove the need for complex runtime control. In contrast, performing interpretation as runtime requires us to consider a number of control issues. If we treat the surveillance problem as an on-line process we need to describe how it can use its understanding of current scene context to construct an evolving understanding of scene activity. On-line reasoning would be useful in real-world situations, for example, the system could be connected to a recording device to selectively record noteworthy segments with annotation for subsequent analysis. In this section we describe the problems raised by this requirement in terms of how the system's operation can be controlled. We briefly describe previous work that appears suited to solving this part of the surveillance problem and identify those aspects which still remain to be addressed.

One of the central objectives of a surveillance system is to develop an understanding of what is happening in the scene under surveillance. This objective is made more complicated by (1) the possibility of different explanations for the observed evidence, and (2) the amount of evidence that is continuously supplied to the system observing a dynamic scene.

These two problems do not fit well with conventional approaches. The difference between off-line and on-line reasoning is made more apparent if we consider how it affects the use of task knowledge. If we think of tasks as being discrete entities with no side-effects then we can have a lattice of all possible subsets of tasks. The tasks in the lattice are designed to enable questions related to the application domain to be answered. The script-based approach performs all tasks because at runtime we do not know which questions will be

---

4In chapter 5 we show this assumption to be false. A task can side-effect another task, which may make some subsets invalid. Here we will just consider the ideal case.
asked. In the situated approach the system performs according to any single subset of tasks in the lattice, because the question is known at runtime.

We separate the review into two parts addressing: first, the relationship between the observer and its environment, how context affects observation, and how attentional mechanisms have been developed; following this we describe approaches used for planning and control.

4.2 Observation and the Environment

An important influence upon our understanding of the environment and the use to which it is put comes from our social context. We will only briefly discuss this subject here with reference to the work of Garfinkel (see [81] and also Heritage [97]), whose objective is the recognition of activity in a social context. He has performed experiments [81, pages 40-44] that demonstrate the presence of "normal" behaviour or "maxims of conduct", that Heritage [97, page 117] has summarised by saying that:

Definition 2.2 Reflexive accountability exists when each of the participants hold one another accountable. This social world will exist when the following three conditions hold: (1) that the social participants are "aware" of the norm, (2) that they are, on occasion, capable of reflexive anticipation of the interpretive consequences of breaches of the norm, and (3) that they attribute the conditions (1) and (2) to each other.

It is debatable whether the official-observer is part of the social world since it does not communicate with the participants in the environment, however, knowledge of the norm is necessary to understand what is taking place in the social world. The norm becomes most apparent when it is breached (or "broken" in the sense of Heidegger [256, pages 36–37]). Heritage [97, page 116] provides an illustrative example where a greeting is not returned. Post-mortems performed on this breach show how explanations for its cause are excuses or justifications. This reflexive accountability of the actors keeps perceivedly normal conduct "on the rails" because they are able to anticipate some of the interpretations that their exercise of the options will give rise.\(^5\) Although not covered here, this approach would be useful for incident detection where the "breaking-down" of expected behaviour would assist in the description of observed deviation from the norm.

One approach for expressing this social context is given by Schutz's [212] (also see [97, page 54]) idea of a "common world". This is based upon the intersubjectivity of the agents with two idealisations (1) the interchangeability of standpoints, and (2) the congruency of the agent's systems of relevance. Garfinkel reinterprets this in his "documentary method of interpretation" where an actual appearance is taken as "standing on behalf" of a presupposed underlying pattern [97, page 84]. For example, in a conversational exchange the participants do not treat utterances literally, instead they are understood by reference to unspoken assumptions and presuppositions that each party attributes to the other. This is an example of the role that context plays in interpretation, and the importance of having a

\(^5\)In the same way as Gibson's "valence" described in section 3.4.
“common ground” of mutual understanding. In chapter 5 we use a “typical object model” to provide the set of background assumptions against which interpretation takes place.

### 4.2.1 Routines

Garfinkel’s work provides a framework in which to describe routine and situated behaviour. Most activity is routine in nature, being that regular, practiced and unproblematic activity that makes up most of everyday life. Much of the surveillance problem involves understanding this routine activity and so it is appropriate to investigate this instead of focusing on the identification of more novel activity, such as collisions, which are better represented as deviations from the norm. In the everyday world these are rare episodes, and although they do occur, are not the main elements of the surveillance problem.

We would like to provide a stable organisation for understanding the social activities of a given surveillance task and, according to definition 2.2, this can be done by giving detailed consideration to the participant’s understanding of their “empirical circumstance”. In the situated approach we proposed approximating this by using simple local models that express normal or typical routine behaviour. The effect of definition 2.2 on this model is that it is unlikely that all the routines used for understanding activity in the road-traffic domain will be the same as those in the office domain, because the routines used by the participants are different. For example, although walking-through-the-office and driving-along-a-road are similar, typing-at-a-keyboard is not. More importantly, it illustrates the problem present when we try to define routines in an implementation since we do not know what the evolving context in which the routine is to operate will be. The best we can do is form some approximation.

Although the official-observer does not itself move, the processes used to interpret what it sees are likely to be based upon those facilities developed during active participation with the environment during everyday life. Braithwaite [21] provides an example of the important role played by the perceiver interpreting the exhibited behaviour of dynamic objects. It is by participation that the perceiver learns what are the important features that need to be attended, and so increases the pertinent perceptive skills required. This is a complex problem (see Lave [138], Lave and Wenger [139], Whitehead and Ballard [252]) and outside the scope of this dissertation. Instead, we have made assumption 8 saying that the program designer can provide the necessary knowledge. However, we still have the problem of how to represent this learnt knowledge (the difficulties of which are described by Dreyfus [63, page zzzz]) both in terms of rule representation and how task knowledge affects rule selection. These are described further in chapter 5.

### 4.2.2 Task

The understanding formed by the perceiver of the observed environment is affected by context. By this we mean that the same state or activity may be interpreted in different ways. A visual example of this is the duck-rabbit figure given in figure 2.12. Interpretation of the figure can be affected by context, for example, if it were given a background
portraying ducks, a duck is the likely interpretation. Changing the background to rabbits would alter the interpretation. A further example is provided in figure 2.13, which illustrates how the same data can evoke quite distinct interpretations and how expectation due to context, that is built on past and concurrent interpretation, affects interpretation of the environment.

We can also change interpretation by giving an instruction. When asked to look for a rabbit in figure 2.12, the figure resolves into a rabbit with ears. Associated with this active interpretation is a change in which parts of the image become more important. Figure 2.14 illustrates this with two examples from the large number of experiments reported in Yarbus [260]. In the figure, parts (b) and (c), show that the scanning patterns of saccadic eye movements are highly selective for the particular task at hand. This special behaviour is due to the human visual system having a small fovea, and its purpose is to quickly move this highly sensitive visual receptive area to different spatial targets. This illustrates the observation reported in psychology textbooks such as Rock [202], that people do not scan a whole scene with the same intensity but rather, after they orient themselves, focus visually on those objects needing attention. The result of this is that we do not see everything, and miss out those things which can be ignored for the task at hand.

We can separate the visual processing into two: (1) peripheral, that uses data of low resolution over the entire field-of-view; and (2) foveal, that has a small area of high resolution that can be moved within the field-of-view. One approach would be to use a pointable spatially-varying sensor. In the surveillance problem we do not directly address this because we are not considering low- and intermediate-level vision. Instead we will use the marker approach introduced in section 3.4. Here we can not change the images that are being collected, we can only select which results from intermediate-level processing to use. Being able to alter the amount of image data used provides the computational advantage of only having to process the relevant portion of the potential image data. The problem has two parts: deciding where to point the fovea, camera, or marker and the selection of which operations to perform on the selected visual data.

This area of research called “active” or “animate vision” (Bajcsy [12, 13], Ballard [14], Tsotsos [240]) concerns how to apply intelligent control to the data-acquisition process that
depends upon the current state of data interpretation, and which provides the opportunity to understand the visual processes in the context of the visual behaviours engaging the system. This task knowledge is used to assist in determining where to look next based on the current state of interpretation and in the development, in chapter 5, attentional characteristics from active vision are adapted for selective processing of the dynamic data.

4.2.3 Attention

Given the results from intermediate-level vision and a task, we need to identify the relevant properties from these results. This process, called “attention”, concerns taking possession of one of several simultaneously possible groups, objects or object parts, where the objects are from either the sensory or intellectual domains. In the sensory domain we use accommodation or adjustment of the sensory organ, and in the intellectual domain we can emphasize a particular aspect of an idea to invoke its associates and so select the route that a train of thought is to take. The reason for this attentional process is the supposed presence of a bottleneck of limited capacity. There are two theories that differ in their placing of this bottleneck at or before perceptual recognition or categorisation takes place. These theories have not been empirically resolved, but see Allport [7] for discussion of this controversy. For more general details see Humphreys and Bruce [109, pages 143–190], Moray and Fitter [164], and the short review by LaBerge [132].

An example attentional mechanism is given by the marker approach outlined in section 3.4, where we described how a marker is used to index the “location” of an object by representing the results of selection in a sensory or intellectual domain. For example, when we determine the position of a contiguous blob of space, perhaps via gestalt like principles. Once we have marked a location, we can bring specialised processing to bear on the target. This may involve things like working out what it is, by first attending to the whole object then adjusting downward to align with parts of the object. This two stage process of first allocating a marker and then using specialised processing, allows associated control issues to be identified at the task- and strategic-layers. This form of
indexical reference was developed as part of Ullman’s visual routine mechanism [243]. A similar form of indexical reference, called “FINST”, is described by Pylyshyn and Storm [191], who present psychophysical evidence that there is a numerical limit of four or five on the number of objects that can be tracked at one time. The ability to track four or five objects is pertinent to the surveillance problem as it can be used to place an upper bound on system complexity, since a human observer can perform surveillance with such limitations. They also separate multitarget visual tracking into two stages, one a parallel preattentive indexing stage and the other a serial checking stage invoked in selecting a response, for example, when we count the number of objects. Miller [160] describes similar limits on the capacity of processing information from other stimuli. Markers present an attractive approach to describing how the observer tracks the dynamic scene objects, and more details of FINST are given in Pylyshyn [190]. We will use the two stage separation of attention described here in chapter 5.

4.2.4 Discussion

The discussion has identified three related aspects of control that operate on control issues: the interpretation of the social world, particularly the behaviour of others; the effect of task knowledge upon the behaviour of the system; and the effect of attention and how selection can be made.

The background on social context, animate vision and attention provide foundations that begin to solve the complications of different explanations and amount of evidence that were identified in section 4.1. These two problems are coupled by the fact that performing one surveillance task means ignoring data that is not related to the task. This cuts down the amount of evidence but also means that were we to process the same input data with two different tasks, then the properties extracted for the reasoning are likely to be different (for example, see figure 2.14). This approach may be problematic and not appropriate to some problem domains (such as passive monitoring), but we claim that this active approach to surveillance better reflects how surveillance is performed. It provides a more appropriate use of a given set of finite resources, that in passive monitoring would be spread evenly but thinly, by allocating the resources to where they are needed.

4.3 Computational Approaches

In the surveillance problem we have an objective provided by the “surveillance task”, for example, identify the cars that overtake, or the people that close doors. The task itself does not require planning, although the operators used to identify what is happening in the scene may help to organise what should happen next.

There is a large literature on planning and temporal reasoning (see, for example, Tate [228] and Allen et al. [6]). In the discussion here we will just consider four approaches that offer solutions to different parts of the surveillance problem. In time maps we describe one approach that enables reasoning about temporal data that has been asserted into our temporal database. In plan recognition we identify the control issues that are present. In
multiagent we describe one approach to modelling local and global data. In embedded systems we consider a framework that addresses most of the issues concerned with a real-time system that operates in the environment.

The result of surveillance is likely to include a description of the history of perception, however the form it takes greatly affects system operation. The options range from remembering all observed events and episodes (as in time maps) to remembering no past episodes but only those that are taking place at the current time (as in embedded systems). In the first instance we can go back and correct our recollections if our interpretation made at the time is later found to be incorrect. In the second case there is uncertainty attached to our interpretations that needs to be represented in our history mechanism. The example surveillance tasks given above do not require an extensive history mechanism.

4.3.1 Time Maps

An early intention in the VIEWS project [108] was to use the Time Map Management system (TMM) developed by Dean and McDermott [56]. A TMM is used to efficiently reason about logical propositions whose status changes over time. In one of its simplest forms, given a set of propositions which hold at a given time, a TMM infers the propositions that may hold at a later time, after a sequence of events has occurred (see Dean and Boddy [55]). The propositions are assumed to persist in the absence of relevant change, where changes are the result of events whose effects are described by means of causal rules. A causal rule may indicate, for example, that after a vehicle enters onto the roundabout from an entry road that it is no longer on the entry road; another rule may say that if the vehicle entered the roundabout at time \( t \) that it will travel on the roundabout for at least some minimum time \( T \) and exit the roundabout at some time \( t + T + \Delta \); and so on. In the simple case described, the operation of the TMM is straightforward. The algorithm starts at the time for which it has complete information, and moves forward along the time axis looking for causal rules which may be triggered. If found, all applicable rules are inspected and the status of the temporal database updated accordingly.

The problem with using this approach is obtaining all the necessary data and then using the TMM to perform predictive reasoning which might be more easily done without it. The TMM is likely to provide a level of consistency analysis (by continuously comparing expectations with observed activity) but at a considerable run-time cost.

4.3.2 Plan Recognition

The compositional nature of the script-based approach described is section 3 lends itself to plan recognition. A review and formal treatment of plan recognition is given by Kautz in Allen et al. [6, pages 69–125], where its origins in story understanding (Schank and Abelson [209]) are described. Other approaches to this problem include Schmidt, Sridhara and Goodson’s [210, 220] BELIEVER system, which interprets linear sequences of intentional statements about people in everyday environments. This has similarities to the adaptive planner of Alterman [8]. Plan recognition would be suitable for a script-based approach,
and could be implemented using a TMM (see Dousson et al. [59] for example), and the
uncertainty present from visual input could be addressed by using the Bayesian approach
described by Charniak and Goldman [35]. However, this suitability for off-line reasoning
reduces its suitability for on-line control. The approaches are better at explaining what
happened than being used to understand what is happening in the here-and-now.

4.3.3 Multiagent

There is a partial overlap between the surveillance problem and multiagent planning (for
e.g. the PHOENIX system of Cohen et al. [41]), in that the official-observer is
reasoning about the dynamic processes that it observes in the same way that an agent is
reasoning about the other agents that populate its world. However, we are not dealing
with cooperative or adversarial agents, but just the observation of these agents as they
carry on their everyday lives, and a key difference is that in the surveillance problem we
are not able to communicate or control the observed agents.

Lansky [134] presents an example of a multiagent planning theory which uses the Group
Element Model (GEM). GEM exploits causal independencies by using explicitly defined
constraints to synchronise plans. The process of plan synchronisation is not limited to a
strategy of planning how to achieve each separate component task and then combine the
results. Instead, a more general, adaptive strategy is used that can bounce back and forth
between local (i.e., single-agent) and global (multiagent) contexts, adding events where
necessary for purposes of synchronisation. Lansky describes how these planning loci can
both overlap and be composed hierarchically. The use of local and global contexts would
allow the official-observer to both reason about a single-object and see how this reasoning
fit into the global context. However, much of the supporting mechanism (see [134] for
details) seems unnecessary for the surveillance problem because we are not planning the
activity of the objects that we observe.

4.3.4 Embedded systems

An embedded reasoning system is one that is situated in the world and which operates
effectively given the real-time constraints of its environment. Georgeff and Ingrand [84]
describe an embedded reasoning system called the Procedural Reasoning System (PRS)
which uses means-ends reasoning to govern future behaviour. PRS explicitly represents
attitudes of belief, desire and intention, allowing them to be manipulated and reasoned
about, providing complex goal-directed and reflexive behaviour. This framework would
allow the official-observer to consider its own goals in addition to those of the scene-
objects that it perceives. The system consists of a database holding current beliefs and
facts about the world, a set of current goals to be realised, a set of procedures or rules,
and an interpreter for manipulating these components. At any one moment, the system
also has a process-stack containing all currently active plans, which can be viewed as the
system's current intentions for achieving its goals or reacting to some observed situation.
The rules describe how certain sequences of action and tests may be performed to achieve
given goals, how to react to particular situations, and also includes meta-level knowledge that enables manipulation of the system's own beliefs, desires and intentions. PRS comes close to solving most parts of the surveillance problem, however, it is not really situated in the way described in the section 3.2.4 and appears to be more closely associated to plan recognition.

To provide a more situated implementation we could adopt Agre's [2] "RUNNING ARGUMENTS". This technique is difficult to describe because there are at least two intertwined theories at its core. The first is to do with planning, which we describe here, and the second is about the separation of program components, which we describe below in section 4.3.5. The RUNNING ARGUMENTS technique does not develop plans as such, although they do exist in the form of hardwired "action-descriptions". These action-descriptions are written in a language called "MACNET" (see Chapman [34] or appendix E for details) that expresses what the program is to do given the data of the current and previous clock tick. The rule based form used to define the action-descriptions makes a comparison to the standard production rule form inevitable. The main difference is in how conflict resolution\(^6\) is addressed. In the RUNNING ARGUMENTS system conflict only occurs when two or more "proposals" try and "fire" the same operator, and any occurrence is resolved by assigning rule precedence to the rule definitions. This approach allows a number of operators to be "fired" on each clock tick as opposed to the usual one per clock tick in production systems. This language is not suitable for planning, but can be used to describe plan like elements called "routines".

Rosenschein and Kaelbing [204, 205] present a more complex representation language called "REX" that is similar to MACNET in that it too is compiled to provide a resultant combinatorial logic circuit. The REX language is attractive because it has a basis in a formal logic that is similar to that of Moore [163], however, the implementation details provided in [205] are difficult to understand. In [204], Rosenschein describes how an additional layer of compilation can be added to enable proof correctness to be performed upon the supplied rule specification. This involves re-expressing the rules as clauses in a new language \(\mathcal{L}\). The rules are now generated as a side-effect of performing a proof analysis on the clauses written in \(\mathcal{L}\), with this proof analysis indicating the completeness of the specification, at the cost of an additional layer of compilation. This approach would begin to address assumption 8 we made in chapter 1, but does not in itself necessarily provide a situated implementation. The reasons for preferring MACNET over REX are that it is better described and that being part of RUNNING ARGUMENTS it would be a proven choice should RUNNING ARGUMENTS be adopted. The development of a new representation language was not an objective of this work.

Even if we replaced the procedural rules in PRS by MACNET or REX, we would not change very much, and would still not have a situated approach. Next we investigate the second part of RUNNING ARGUMENTS.

4.3.5 Modularity

In our description of the official-observer we have not yet considered the effect of cognitive architecture on planning and control. Fodor [70] describes the traditional separation made in cognitive science between input/visual/peripheral systems and the central system. This view is not held by all researchers, for example, Brooks [23, 24] provides a different view that uses an orthogonal separation based on task-achieving behaviours. Agre [2] and Chapman [34] are both proponents of Fodor's input/central split, (see their descriptions for further details). A brief description follows. On the input side we have a collection of perceptual and motor processes, each of which are to a large part innate, localised to specific brain areas, and task- and domain-independent. Each element of this collection is a module of the input system. Fodor argues that the central side is different, saying it is not modular, being instead a single homogeneous central system. The justification for this is that anything you know can potentially be used in any cognitive task. Agre uses this split in RUNNING ARGUMENTS, with the “central-system” holding the rules and the “visual-system” holding a collection of information gathering operators.

The visual-system is based upon Ullman’s argument [243] for the integration of multiple visual operators that perform particular sorts of perceptual work such as tracking (which we introduced in section 3.4), representing shape properties and spatial relations. Previous work on operators also includes: Romanycia’s [203] description of a programming language that uses visual operators to commute properties and relations present in 2D images of simple geometric shapes; Mahoney and Ullman’s [149] description of low-level visual operators that operate on more complex shapes and curves to identify “image chunks”; and Chapman’s [34] description of visual operators used in a video game context. No one has used visual operators in a natural task domain with access to camera image data in the way described by Chapman (such as those which use “activation planes”7). However, there is related work, including that on tracking described in section 3.4, and work on “active contour models”8. Chapman’s work can be seen as an initial step towards the objective of defining a set of appropriate visual operators for use in real world application domains. On their own, these visual operators do not do much, but combined via rules held in the central-system, enable the system as a whole to respond to changes in the “world”.

The central system contains rules of the form described in section 4.3.4, which are used to select when an operator is to be used and what arguments are to be supplied to an activation. Crafting the rule-operator pairs into sequences (constructing routines) is done by making the result produced by one operator fulfill the input requirement of the next rule. However, this is not the only way a particular rule can be fulfilled, thus allowing the mechanism to react to similar situations that arise via a different route. A

---

7Activation planes are used to keep track of interesting regions of the image, as in Ullman’s [243] routine for computing containment.
8Active contour models have been used to define outlines (Kass et al. [116], Cohen [40]) and dynamic regions distinguished by a particular visual property, e.g., texture and/or colour (Ivins and Porrill [111]).
routine provides an abstraction for a common pattern of interaction between an agent's central-system and visual-system. This reduces "planning what to do next" to a matter of deciding what to do "now" based upon how the world is "now". Only the operators have access to the "world" data structures and the central-system only receives the results of the operators. This allows the central-system to use a simplified description of the world, that only needs to have the information necessary for making its action-selection. This restricted state ensures that the system can only reason about the current situation.

This separation of input and central system and the tight coupling between them provides one possible foundation for the situated approach, allowing us to address control without traditional planning or plan recognition. Although good for reasoning about the official-observer's ongoing circumstance, in the surveillance problem we also need to reason about the moving scene objects as we described earlier in section 4.2.

4.3.6 Knowledge organisation

The organisation of knowledge about observed properties was mostly dealt with in section 3. Here we are considering this organisation with the objective of controlling the reasoning performed. The ALVEN system (Tsotsos [237]) has similar objectives to the surveillance problem, it is passively understanding the difference in visual motion between pre- and post-operational film sequences taken of the left ventricle. Tsotsos generalises this cardiology example in [239] and considers the basic capabilities that should be present in an attentive vision system that is addressing time-varying phenomena. The ALVEN project uses both atemporal and temporal control, which are integrated via a semantic network. This semantic network is a development of the one proposed by Badler (and described in section 3.2.1). The atemporal control includes goal-directed, data-directed and model-directed inference mechanisms with the claim that each compensates for the deficiencies in the others. The temporal control uses a modification of relaxation labelling, called a "temporal cooperative process", to accumulate and integrate the dynamic information. ALVEN uses a rich search dimension to enable and distinguish search in image space from search in hypothesis space. Within these search spaces, focus-of-attention can be generated and maintained. Tsotsos [238] notes how this control framework can support attentive vision by using Ullman's visual routines [243], providing an attentive foundation for the situated approach. However, these knowledge based approaches are typically less uniform and more complex in managing the control and do not address the amount of evidence problem given in section 4.1.

4.3.7 Uncertainty

Task knowledge needs to take account of the uncertainty present in perception. One solution is to use Bayesian inference, which is described in detail by Pearl [183] and Neapolitan [172]. Bayesian networks integrate a mechanism for inference under uncertainty with a secure Bayesian foundation. They have been applied to various research domains such as medical diagnosis (Spiegelhalter [219]), and model-based vision (Agosta [1], Levitt et al.
(145) which follow the approach described by Pearl and Neapolitan where once constructed the nodes and links do not change over time. This "structurally static" approach involves determining the graph structure of the network, and then supplying prior probabilities for the root nodes, anticipatory values for leaf nodes, and conditional probabilities for other nodes. The inference algorithm is then run for each addition or retraction of evidence from the leaf nodes. Breese [22] describes how to dynamically construct a structurally static network, an approach that has been applied to natural language understanding by Charniak and Goldman [35, 87]. Dynamic construction allows a smaller task specific network to be built, rather than a large general purpose one.

Some researchers have extended this approach by applying it to dynamic domains, where the world changes and the requirement is to reason over time (see for example Kaelbling [119]). Applications include sensor based mobile robot navigation and target tracking by Dean et al. [54], and monitoring light beam sensor data by Nicholson and Brady [176, 177]. The dynamic nature of the domain is captured by extending the network over time, adding a new time-slice of nodes, at each clock-tick. These dynamic networks are Markovian, which constrains the state space and limits the history maintained by the network.

Rimey and Brown [200] do not use a dynamic Bayesian network, although their system is used in the dynamic domain of active vision. Their system called "TEA" uses specialised Bayesian networks together with a maximum expected utility decision rule that it runs iteratively to select the evidence gathering actions that maximise an expected utility criterion. TEA uses three different forms of specialised Bayesian networks that comprise: (1) a "part-of net" that models subpart relations, (2) an "expected area net" that models geometric relations, and (3) a set of "task nets", one for each task that produces actions which are unique to the task.

All these applications of Bayesian networks to dynamic domains are relevant to the surveillance problem, with the emphasis of on-line reasoning being attractive for the situated approach. We essentially used dynamic Bayesian networks with embedded approach in [107].

4.4 Summary

We have covered a range of approaches to controlling the operation of a surveillance system. We are mainly considering the case where control of the system is seen as an important issue for performing a particular task. We have identified that this form of task dependence is only relevant in the situated approach. In section 3 we identified the use of markers and deictic representation as being necessary should we want to provide spatial reasoning that is related to an observed object's intrinsic frame-of-reference. In this section we have covered some of the control issues raised by this approach, identifying the usefulness of Ullman's visual routines [243], which has been integrated into larger control frameworks by both Agre (see section 4.3.5) and Tsotsos (see section 4.3.6). We are not just addressing attentional control. We require a framework that also allows conceptual reasoning about
the behaviour of the observed object. This capability seems to be partly provided by RUNNING ARGUMENTS and PRS (described in section 4.3.4). We develop this situated approach in chapter 5, where we also use Bayesian networks to address the problem of uncertainty present in high-level visual understanding.
5 Conclusions

The reviews in this chapter have covered the wide range of research necessary for addressing HIVIS. We have separated both the categories of knowledge, and how this knowledge addresses the surveillance problem. We also identified two approaches, called “script-based” and “situated”, to the surveillance problem. These reflect the difference between traditional passive vision research and more recent work on active perception. We have also illustrated the role that knowledge held by high-level vision can play in image understanding.

An overall objective is the development of an approach that places emphasis on spatial representation and reasoning. The cellular topology was found to provide a useful foundation and we will use this in the development of a spatial database that is needed to hold information about the static environment. This requires providing a suitable structure for the spatial database together with the functionality required by the runtime system to access this database. This is developed further in the following chapter.

The script-based approach can be given simple task- and strategic-layers, allowing us to address the issues of implementing and testing a spatial representation without addressing control issues. The main use of the spatial representation is for providing contextually indexed information about moving scene objects. This requires the development of a theory of events and episodes that can be used to reason about observed behaviour. To do this we will use cellwise time (introduced in section 3.3) to provide properties that can be asserted in a temporal database and reasoned about. This offers a good option for our first implementation, because there are well defined “traditional” approaches, but it does not fully address the surveillance problem.

In comparison, the situated approach is less well defined, having more complex task- and strategic-layers, because of connections between social context and vision research. Human motion understanding has a long history (see Johanson [112], Bruce and Green [26, pages 321–333]), but here motion information is just a starting point and we are focussed on understanding what the moving objects are doing (see Newtonson [175] and From [77]). This second implementation shifts emphasis from spatial representation and reasoning to the provision of situated control, but this is only to allow deictic representation to be supported, which offers a more natural form of spatial reasoning. This “here-and-now” quality of situated control is important for identifying when some predefined situation is seen to be occurring, so that the observer can attend more closely during the actual occurrence of the situation rather than using off-line processing to identify what happened when the occurrence has past.

The work in this dissertation seeks to develop a prototype (see assumption 1) that satisfies the surveillance problem as stated earlier, something that is not done by the work we have reviewed. In the sections on spatial representation, events and behaviour, and control and planning, we have identified the foundations that we will extend, integrate and improve upon in the following chapters.
CHAPTER 3

SPATIAL REPRESENTATION

The analogical representation of space extends the common spatial representation developed in chapter 2 section 2.3 where we introduced the foundations of the spatial primitive called "cell" and the regular cell complex. In this chapter we describe how regions of space are composed from the cell primitives. "Regions" are used to represent physical and semantic properties in the environment. This environmental data is given in the form of ground-plane scene geometry, which is part of the application specific knowledge. Other types of domain knowledge include: expected object behaviours, allowed routes, and collateral knowledge.

We are using a description of space that provides an analogical representation of an outdoor or indoor scene and the objects that move through that scene. This is primarily because "contextual indexing" in this kind of representation can provide a model of space that scales well with scene complexity. This representation supports reasoning about spatio-temporal data, which is used to detect predefined events.

We choose the cellular topology because it provides a mathematical foundation upon which to build. The topological 2-cell is amenable to different implementation techniques (we describe both raster and vector approaches that require no change to the representations built upon the cellular topology, removing the need for a costly translation stage) and, as described by Fleck [68], it can also be used to represent the temporal axis. This flexibility, due to abstraction present in the formalism, allows us to separate implementation details from the computational model, so that we can concentrate on the essential features first and fill in the low-level implementation details later.

In addition to the development of regions, the extensions concern the development of a database to hold the spatial domain knowledge in a form that allows it to be appropriately accessed at runtime. This chapter begins with a description of the computational model, consisting of cell space and the hierarchical-database, and is followed by implementation details and an example.

¹This chapter is based the journal paper "An analogical representation of space and time" (Howarth and Buxton [105]).
1 Cell space

We do not deal with cells directly, instead we use an abstraction based upon cells called "regions", which are present because they ease the expression of spatial data. Regions are used to express features of the ground-plane scene geometry such as physical and semantic environmental properties, which are part of the application specific knowledge. To obtain this knowledge, we have used: video footage of the scene in question and introspection about the everyday routines. The road traffic domain is particularly well suited to this analysis having additional data concerning our everyday knowledge of driving and perceiving vehicles on the road as well as constraints of the road system and traffic laws. Part of this behavioural information can be expressed as static knowledge concerning areas where behaviours typically occur. In the road traffic domain we can note the locations where cars turn, where they travel in a particular direction, where they give way to other traffic, etc. In the office environment, we have typical routes of travel, for example, to doorways and desks, and the area near the laser printer where people collect print-outs. This information needs to be captured in the spatial representation, so that an interpretation of each observed object's behaviour can be made from its position in the scene. This static scene knowledge has a spatial extent, i.e., is only applicable to a defined area of the scene and provides semantics for the scene. To capture this idea we will define a spatial primitive "region":

**Definition 3.1** A **region** is a (closed) 2D area of space composed from a cluster of cells. The spatial extent of each region is specified by some property that is continuous throughout.

The cells that form a region are part of the regular cell complex that provides the structure for the underlying space. In appendix B section 2 we give a more formal development that shows how a region is a subcomplex of the regular cell complex, and that a subcomplex is a regular cell complex in its own right. The definition of region provides an encapsulation of the space it represents, forming a single symbolic entity. Being able to deal with regions in this way is intuitively pleasing. If we look inside the encapsulation we find that the region is made up of cells, the structure of which is defined by how the space has been structured (i.e., the regular cell complex). The boundary cells of a region are different from the rest of the cells that make up the region; they delimit the region. The boundary cells can be thought of as the skin enclosing the contents of a region. The cellular topology solves the problem of boundary ownership (see appendix B section 2.3) although care needs to be taken in the implementation. The cellular topology also provides some rules about how cells can be joined together (i.e., cell and region adjacency) which, when followed, help reasoning about the connectivity of space (e.g., neighbour relationships).

We have now described the topological representation that acts as the foundation for the analogical representation of space used to perform high-level reasoning. In the next section we describe how space is structured into regions and how these regions can be stored in a database.
Chapter 3. Spatial representation

2 Hierarchical database

To represent (1) the structure of the environment and (2) the properties that are attached to the spatial model of the environment, we use two forms of region, called "leaf" and "composite", the names are to reflect their position in the hierarchical database that they are used to construct. The main purpose of this database is to provide spatially related contextual information about scene properties that can contribute towards describing the activity of the dynamic objects that occupy that part of the scene. We call this information gathering "contextual indexing". To provide this functionality, various forms of contextual information are stored in the database covering spatial properties like "room", "walls", and "floor".

2.1 Leaf and composite regions

We use the two forms of region, leaf and composite, to separate the explicit representation of spatial extent from the various attributes that can be attached to a space. We do not attach attributes to leaf regions, they are used to provide an interface to the underlying cellular topology. Leaf regions are the most primitive spatial database element, they tile the ground-plane map, are not allowed to overlap, and describe the smallest regions in the database. New leaf regions can only be added to the ground-plane map by subdividing a current leaf region, by use of the leaf-split operator. Leaf regions can only be removed by forming the union of the leaf region to be removed with another neighbouring leaf region, to create one continuous leaf region (i.e., space cannot be removed) by use of the leaf-merge operator. Both of these operators alter the structure of space.

In comparison, each composite region’s shape is defined by what it represents, be it geographic (e.g., a room) or behavioural (e.g., a typical path through the room). Composite regions are composed of leaf regions which have been joined together to form a continuous space by a union operation called composite-union (so that the space inside the region has no boundaries but has the structure of the unioned-together regions). Conceptually, composite regions are treated the same way as leaf regions and can be used to build other larger composite regions. Composite regions overlap when they express spatial semantics that share the same space. By using the composite regions in this way, we have formed a hierarchical database, an example of which is given in figure 3.1(a). The example in this part of the figure illustrates the separation between leaf regions and composite regions, notice that a leaf region can be used by zero, one or more composite regions. Part (b) shows the ground plan of the room shown in figure 2.5, to which we have added typical paths through the room (see part (c)) and which are shown separately in (d). The three top most nodes in part (a) denote the three paths.

The various components in the room can be related in a hierarchy that is based on enclosure (or part-of links). Parts (e) and (f) of the figure illustrate the spatial decomposition of the office area. At the top of the hierarchy we have the whole room, which is decomposed into the walls surrounding the floor space and the doors that afford entry and exit. The floor space is decomposed into (1) the table area that while present block
access to this space (we assume tables to be static during a scenario), and (2) the walkable area which can be used by relocating easily moved objects. The walkable space includes chairs and a clear area (usually free of chairs which are placed in front of the computers on the various tables). The clear area has the typical paths of travel used by people walking through the room and doorways. Figure 3.1(a) only shows a subpart of the full set of leaf regions that can be generated from the spatial-decomposition shown in part (f). In effect, the leaf regions are structuring the space used by the composite regions. This means that we only have to store a full spatial description of the leaf regions. Here we have formed the set of composite regions, $CR$, the set of leaf regions, $LR$, and the set of relationships that hold between these two, $\{ \langle lr, cr \rangle \mid lr \in LR, cr \in CR, child-of(lr, cr) \}$ and between different elements of the composite regions $\{ \langle cr_1, cr_2 \rangle \mid cr_1, cr_2 \in CR, cr_1 \neq cr_2, child-of(cr_1, cr_2) \}$. These relationships are defined by the application domain specific child-of predicate and provide the $\langle child, parent \rangle$ links in the hierarchical database. A formal description of how leaf-split, leaf-merge and composite-union operate is given in appendix B section 3.

We are using two types of connectivity, the first is inter-region connectivity which is the connectivity at the region level such as region neighbours, and the second is intra-region connectivity which is the connectivity of the spatial elements inside a region. The separation of leaf and composite regions means there are two forms for each type of connectivity.

- Leaf region intra connectivity, which concerns the implementation chosen for the cells that make up a region.
- Leaf region inter connectivity describes for each leaf region its neighbouring leaf regions.
- Composite region intra-connectivity which is described by the child-link relationship in the hierarchical database.
- Composite region inter-connectivity is the neighbour relationship that holds between composite regions. These neighbour relationships form a graph, which can be given directed arcs and/or type information.

The cellular representation also uses continuity reasoning in the formation of each composite region, by using function smoothness, to determine the spatial extent of the property a region expresses, i.e., the given property is true for all sub parts of the composite region.

2.2 REASONING WITH REGIONS

Once we have this hierarchical database of regions, we then need to identify how to use its contents during runtime. The cell structure in each leaf region is described by an adjacency set of cells (see sections 1.4 and 3 in appendix B). Since sets are being used, set operators can be applied to the adjacency sets that represent regions. Given the set operations for intersection, union and equality, it is possible to determine if one region is contained inside another. For example in figure 3.2(a), the set of cells for the circle is $G$, and the set of cells for the triangle is $T$. $G \subset T$ because $G$ is enclosed within $T$, i.e., $G \cap T = G$ and $G \cup T = T$. Figure 3.2(b) shows the two distinct regions, $G$ and $T$, with each region representing one conceptual entity in its own "plane", and how each region is embedded in a space such that all regions share the same origin, metric and coordinate system. There are two interpretations of this. (1) The static case, where we are representing these two regions in terms of composite and leaf regions, which is shown in figure 3.2(c), and would allow the set operations to be performed in symbolic terms, i.e., $G \subset T$ when $(\forall g \in G \land g \in LR, \exists t \in T \land t \in LR \mid g = t)$ is true, where $g$ and $t$ are cells. (2) The dynamic case, where we use adjacency set functions to access the region-database, to determine whether an object occupies a region. We can reinterpret figure 3.2(b) as being a snap shot of circle $G$ travelling across a 2D plane, such that we want to determine which region it is inside. In this case the static symbolic test can not be applied because now the circle is not a static region. Instead we use the cell structure information held in the adjacency sets, that are allocated to both the leaf regions from which $T$ is composed and
the dynamically created regions for $G$. The dynamic scene objects are represented using
regions that are implemented in terms of adjacency sets. This dynamic construction of
adjacency sets is also used to find out if the paths of two objects cross and to identify
when one object is local to another.

2.3 CONTEXTUAL INDEXING

The description of space given here provides an analogical representation of the outdoor
or indoor scene and the objects that move through that scene. We call this spatial repre-
sentation "analogical" because it has a *homomorphic* mapping to the optic array. In this
sense there is a correspondence between the data contained in the ground-plane projected
analogical representation and the data contained in the optic array. This correspondence
provides the option of using the analogical representation to reason in the ground-plane,
image-plane or both.

This characterisation of the real world provided by the analogical representation makes
explicit the behavioural data that is usually implicit. The data concerns typical object
behaviour that has a local spatial invariance in relation to the environment, and denotes
the likelihood of a particular behaviour taking place at a location. This behavioural
information is most apparent in man-made environments, such as buildings and roads.
The background to this approach comes from the observation that different activities take
place in different habitats and that performing the same activity in different habitats gives
that activity different meaning. For example, the rooms in a house are typically associated
with a particular task (e.g., cooking takes place in the kitchen).

Contextual indexing involves accessing this pre-compiled information to provide con-
textual cues for the formation of conceptual descriptions. The analogical reasoning uses
this contextual indexing of the spatial knowledge to provide a behavioural interpretation,
which describes what the objects are doing in the scene. For example, in figure 3.1 a sta-
tionary person in a chair area is likely to be sitting and working at a computer. It supports
reasoning about spatio-temporal data, which is used to detect predefined events. Context-
tual indexing scales well with scene complexity, both in terms of more complex geographic
scene descriptions, and the number of scene objects (which we describe in chapter 4 sec-
tion 2.3). In the geographic case we are referring to the effect of adding leaf and composite
regions. The identification of a particular leaf region by position is $O(|LR|)$ and the access
of composite region information via parent links is dependent on the distribution in the
tree, but is less than $O(|CR|)$. In section 3.6 we describe a leaf region look-up technique
that makes position location less dependent on the number of leaf regions.
CHAPTER 3. SPATIAL REPRESENTATION

Map file format

| XP | Pann float float float          | 3D point defined by three floating point numbers |
| XL | Lann Pann Pann                 | two points define a line                        |
| XR | Rann Lann ... Lann            | three or more lines define a leaf region         |
| XR | Rann Rann ... Rann            | one or more regions define a composite region    |
| XA | Rann attribute-index value    | assign value to given attribute of composite region Rann |

Map file example

| XA R222 Long-Name "Roundabout South Cycleway" |
| XA R222 USED-BY CYCLE |

Table 3.1: Map file format and example.

3 IMPLEMENTATION

The theory described in section 2 and chapter 2 section 2.4 has been implemented in two versions of a program called "SPATIAL-LAYOUT", which has been written in Common Lisp and tested on a Sun-4. The two versions differ in the cell shapes they use to express leaf region intra connectivity. In the first version a regular square cell shape is used, having a clear relationship to an occupancy grid (described in chapter 2 section 2.4.1), which was the reason that this cell structure was initially chosen. It served to simplify the implementation, but it was, unfortunately, memory store intensive. To address this the second implementation used a triangular tessellation of each region, together with suitable changes to the cellular set operators. Before describing the merits of these two versions we first discuss how the polyhedral map information is acquired, and describe the general framework of SPATIAL-LAYOUT.

3.1 MAP INFORMATION

The application domain specific map information contained in SPATIAL-LAYOUT at runtime is obtained by inputting by hand all the geometric and behavioural information required for each new static scene. To assist with this time consuming knowledge acquisition a program called "MAP-EDITOR" was developed by members of the VIEWS project. The MAP-EDITOR is used to create a "map file" that contains geometric data in the form of polyhedra (see definition 2.1) described by points and lines, to provide leaf regions, which in turn are used to define composite regions to which attributes can be attached (as described in section 2.1). The basic format and a short example is shown in table 3.1. An alternative map file format is described in appendix B section 4 that uses a type lattice based on composite region attributes.
3.2 System Overview

The SPATIAL-LAYOUT program contains the four strata shown in figure 3.3, called database, regions, cells, and implementation. These strata are relatively independent and support precompilation. The cells provide the appropriate level of detail for the construction of regions and from these regions, the database that is used by the access functions.

The structure of the database is given in figure 3.4 which shows the data structures used to store the key elements. The leaf-outline slot is just for display purposes, so that we can draw each leaf region, and the composite-attributes slot is used to hold contextual information about each composite region. We describe the use of the sbound slot in section 3.6.

The figure 3.5 shows the two stages of SPATIAL-LAYOUT life cycle, called "construction“ and “runtime“. The construction stage takes the map file and generates the structure that holds the spatial data so that it can be accessed by the runtime system. This structure holds the leaf and composite regions, and attributes as shown in figure 3.4. The construction phase also adds sbounds to make accessing the regions more efficient. Once constructed the static database is not altered by the runtime access functions.
Chapter 3. Spatial representation

Next we discuss the two forms of cell implementation that have been used in SPATIAL-LAYOUT. This cell implementation concerns the representation of each leaf region. As illustrated in figure 3.2(b), each leaf region is held in its own separate plane, an instance of the problem of representing either the foreground or background of a 2D silhouette.

3.3 Square cell

The first job is to convert the polyhedral description into regular squares. This problem is solved by the scan-line algorithm described by Foley and Van Dam [71, pages 456–460], which makes use of Bresenham’s line drawing algorithm. The size of the cells used is determined by a resolution value, \( g \), which defines the granularity of the structure of space.

Using regular square cells allows different cell types to be expressed explicitly, for example, the filled-in regions present in figure 3.6 are specified by a cell-wise description, given that each cell can have one of fourteen states shown in figure 3.7. The cell types \( b \) to \( m \) are used to describe the boundary of a region, allowing the edge cells of two adjoining regions to both occupy the same coordinate position without intersecting. This means that dense boundaries around a region of space can be formed. This is important to the formation of boundary models as it prevents there being a gap between two neighbouring leaf regions (because the space is locally real). In appendix B section 5 we describe the cell parser that converts the result from Bresenham’s line drawing algorithm.

Figure 3.8 shows a one-dimensional boundary between two regions, giving the cell type interpretation below the cell diagram. At the boundary, both Set one and Set two share a common cell, yet the two regions they represent do not intersect because the respective cells are of types \( i \) and \( g \), so that they can share the same space without intersecting. This model makes clear the extent of each region by supporting a model of space and time with zero-width boundaries. These serve the same purpose as the zero-width time points described in (Williams [255]), namely, to provide a mathematically valid model.

One of the benefits of the regular cell representation is that it uses \( \mathbb{Z}^2 \) with the grid cells just enlarged points. This means that the tessellation of dynamic objects (as in the dynamic case of figure 3.2) uses the same cell structure. This makes intersections efficient and simple with a clear mapping to set operations and parallel implementation. However, this nice cell-level correspondence has both the cost of storage and poor resolution of boundary representation. In most cases this granularity problem is not important, being adequate for most forms of reasoning required. However to address the storage overhead, we investigate the following approach.

3.4 Triangular cell

This approach does not require scan-line conversion and therefore is not dependent upon the resolution value \( g \). Instead we decompose the polyhedral description into irregular triangles (see appendix B section 2.4 for details). Triangles are used to provide a uniform cell level, that has tests equivalent to set union, intersection and equality, that we used in
CHAPTER 3. SPATIAL REPRESENTATION

Figure 3.6: Regions made up of cells. The relationship of this section to the ground-plane is illustrated in figure 3.10 by the smaller of the two rectangular outlines.

Figure 3.7: Cell types: a is an empty cell; b, c, d, e are vertices; f, g, h, i are edges; j, k, l, m are shorthand for combinations of two edges; n is a full cell.

| Two 1-d regions | 
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| A regular tessellation | 
| Cellular interpretation | 
| Set one | 
| Set two | 

Figure 3.8: An example one dimensional boundary using cell-types.
the square cell version, and also enables modelling of concave polyhedra. The storage of triangular cells is similar to that for constructive solid geometry with restrictions to 2D, a single triangular spatial primitive, and a single union composition operator. By using a minimum triangulation for each region, and comparing figures 3.6 and 3.9 illustrates that the number of cell primitives needed to represent a leaf region is greatly reduced, making the structure of space topologically less dense. The price paid for this reduction in storage includes more complex set operators, and the loss of some cellular level flexibility (such as being able to flood fill a region).

The structure of space becomes less explicit when using irregular triangular cells, although the computational model is expressing the same mathematical boundary model. This loss of regular structure also means that we lose the correspondence between leaf and dynamic region tessellation because we have removed the cell-level correspondence.

3.5 COMPARISON

These implementations have shown that selecting a different cell structure does not greatly affect the representation or operation of the hierarchical database. The change in cell implementation did not require a change in the region level of representation because the interface between the two makes them independent of each other. As predicted by the theory, any regular (raster) or irregular (vector) tessellation can be used to implement the cell primitive. However, as usual with computational implementations there is a trade-off between the size of memory and speed of runtime computation.

We have just considered two approaches here. Other possible options include using quadtrees and using leaf regions as cells. Amongst other representations, Samet [207] reviews linear, region and line quadtrees, with the line quadtree being an attractive option
for storing cell types. Quadtree cells use recursive regular decomposition as the underlying basis. This is not the same as the decomposition illustrated in figure 3.1, which is irregular and based on a property we want to express by its spatial extent, also it is a category error to confuse the implementation with what is being represented. These quadtree approaches were not followed because triangular tessellation appeared more attractive. Using leaf regions as cells is viable, the intersection of a convex \( L \)-gon and a convex \( M \)-gon can be found in \( \Theta(L + M) \) time, although this raises to \( \Omega(N^2) \) time for star-shaped polygons that both have \( N \) vertices (see Preparata and Shamos [189, pages 272 and 277]). It also makes the distinction between leaf regions and cells more difficult.

In the database we store each leaf region separately in a "plane". The size of the plane is defined as follows. First we need the side lengths of the rectangular area \((p \times q)\) that encloses all the leaf regions, \( p = (\max x_l) - (\min x_l)\) and \( q = (\max y_l) - (\min y_l)\). Here \( x_l \) and \( y_l \) list of all the \( x \) and \( y \) coordinate values from the vertices of each leaf region. The size of each plane that is used to hold a leaf region is \( p \times q \) units which we want to represent at resolution \( g \). We will consider several approaches to storing the leaf regions.

- Using a 2D array or bitmap has a fixed overhead of each plane taking up \( Q \) bits, where \( Q = ps \times qs \) and \( ps = \left\lceil \frac{p}{g} \right\rceil \), \( qs = \left\lceil \frac{q}{g} \right\rceil \). We can reduce this by using a hash-table so that we only need to store \( N \) bits, where \( N \) is the number of black or the used bits. This is useful since the size of a leaf region is typically much less than the plane, i.e., \( N \ll Q \). where \( Q \) is the maximum value.

- Quadtree require a square that is \( 2^n \times 2^n \) such that it encloses \( \left\lceil \frac{p}{g} \right\rceil \times \left\lceil \frac{q}{g} \right\rceil \). So \( n = \left\lceil \log_2(\max ps qs) \right\rceil \). A regular region quadtree is measured in terms of \( M \) the number of nodes in the tree, where \( M \) can be as high as \( 4nN + 1 \) (see Gargantini [82]). If we use a linear region quadtree we can reduce this space complexity to \( R = (3(n - 1) + 2)N \) bits (see [82]).

- A triangulation of a simple polygon, of \( e \) edges, consists of \( e - 3 \) non-intersecting diagonals and \( T = e - 2 \) triangles. This makes the size of each plane dependent on the number of edges present in the polygon to be represented. As shown in figure 3.9, on average each leaf region decomposes into two triangles. The loss of cell-level correspondence makes intersection tests more complex, but these intersections involve fewer cells with each cell intersection, and because it is between two 3-gons, the result can be found in \( \Theta(3 + 3) \) time (see [189, page 272]).

- Although the hash table has size \( O(N) \), in general it will perform less well than a quadtree.

A summary of the size costs are given in table 3.2 in order of decreasing size complexity. In general, region-region intersection between two regions \( Q \) and \( R \) is \( O(N \times M) \) where \( N \) is the number of cells in \( R \) and \( M \) is the number in \( Q \). In practice returning true at the first positive result reduces this worst case time complexity. The cost of region-region intersection just covers the case when we have identified the two regions are to be intersected. In practice we may want to find out whether a dynamically created region
intersects with any leaf regions. SPATIAL-LAYOUT may contain a large number of leaf regions. To efficiently check each leaf region to see if its region-region intersection test is true or not we next describe how to identify which are the most promising leaf regions to test.

### 3.6 Bounding Box

Accessing ground-plane information is made more efficient by enclosing each region within a surrounding bounding box, called an “sbound” (see Cameron [30]). The bounding box’s sides are parallel to the coordinate axes so that all four corners of the bounding box can be represented by a quad-tuple which describes the bottom left and the top right corners. These sbounds are inherited up the tree from the leaf nodes, so that at higher levels a bounding box is formed that encloses each composite region. Also, dynamic regions (from, for example, an object’s pose-box) are given sbounds. Using the sbounds technique increases the speed of working out the region a point (or regions a polygon) intersects with, by acting as an initial coarse check, prior to the more expensive intersection calculation. Selecting regions to check is made more efficient by considering the sbounds as 2D bounding intervals, treating each axis independently and using an interval-tree (Preparata and Shamos [189]) selection mechanism. Intersecting the two results from the x-axis and y-axis interval-trees just involves the appropriate table look-ups, providing the names of the regions to which a full intersection should be applied.

Cost of accessing a node is dependent on the number of leaves held in each interval tree as described in appendix C section 2.1. The cost of finding the set of leaves A that intersect a given sbound S is dependent on the number of leaves N (i.e., \( N = |LR| \)), and the size of the answer for each axis, \( Ax \) and \( Ay \). There are \( N \) intervals in each interval tree, \( Tx \) and \( Ty \), with each having height \( O(\log N) \). The answer A can be computed in \( O((\log N + |Ax| + |Ay|) + 2(\log N + |Ax| + |Ay|) + A \times I) \) where \( (|Ax| + |Ay|) \) is the intersection of the resultant sets from \( Tx \) and \( Ty \). This approach is better than checking all the leaf regions directly which is \( O(N) \) when \( N \times I > 2(\log N + |Ax| + |Ay|) + A \times I \) where \( I \) is the cost of performing one region-region-intersection test. This is true when size-of(S) \( \ll \) size-of(S_M) where \( S_M \) is the sbound of all \( N \) leaf regions i.e., \( S_M = \bigcup \text{leaf-sbound(leaf)} \mid \text{leaf} \in LR \), and which is the case we are interested in.
3.7 Access functions

The access functions are used at runtime, as shown in figure 3.5, providing an interface between SPATIAL-LAYOUT and the HIVIS-based programs. There are a collection of access functions that includes one for each of the slot names held under the two region types shown in figure 3.4.

There is a further access function called “regions-a-cell-is-in” that takes an adjacency set (AS) description and returns a list of leaf regions that the given AS intersects (i.e., regions-a-cell-is-in :: AS \rightarrow [LR]). The given AS could be the cellular-representation of a dynamic object’s pose-box (as described in chapter 4 section 1.2.2). The regions-a-cell-is-in test uses the abound lookup technique described above. Once we have the list of occupied regions, other access functions can be used to find out more information about the occupied space.

By using the child and parent links we can find out which leaf regions have a particular type, and what types a leaf region has. The latter is used to obtain the behavioural data that provides contextual indexing. Both the child and parent link functions operate on leaf and composite regions, so we define
\[ R = LR \cup CR \] and use this to describe:
- parentLink :: R \rightarrow [R],
- childLink :: R \rightarrow [R].

Application domain dependent region type (ART) information is held by composite regions and is accessed by regionType :: CR \rightarrow [ART], and leafRegionParentTypes :: LR \rightarrow [ART] respectively.

3.8 Summary

This section has mainly concerned the SPATIAL-LAYOUT program, although we did briefly describe the MAP-EDITOR which is used to convert the ground-plane coordinate data into regions for use by the runtime SPATIAL-LAYOUT program. We also described the four strata that make up the hierarchical database, the representation of regions in this database, the access functions that are used to determine which regions a scene object occupies, and how different forms of connectivity are implemented, such as the connectivity between neighbouring regions. The important component of the implementation is the illustration of the flexibility that the cellular abstraction provides.

We have assumed that this knowledge about the scene can be validly modelled using a 2D ground-plane that remains static. These assumptions may not be true in all situations, and the provision of information about surface curvature may be needed, to describe things like road camber. Extending this 2D approach to 3D should not pose to many problems as Paoluzzi et al. [181] demonstrate, although Fleck [68, pages 387-388] indicates that extending the definition of incidence structure (see appendix B section 1.4) beyond the 2D case rapidly gets more complicated because of the increase in possible face relationships.
Figure 3.10: A schematic diagram showing a ground-plane view of the German roundabout. The rectangles denote areas used in other figures, the larger one is the area used in figure 3.15, and the smaller is for figures 3.6 and 3.9.

Figure 3.11: Initial scene analysis.

4 Example

The application domain in this example is the Bremer-Stern roundabout. We are using a ground-plane projection of the image-plane, in which the scene is viewed from overhead to better facilitate reasoning about vehicle interactions, positions and shape. Figure 3.10 shows the ground-plane plan of the German roundabout, pointing out the position of the camera used to take the images in figure 1.4 and the sections of map used in figures 3.6, 3.9 and 3.15. The figure also illustrates how the traffic uses the right-hand side of the road and once on the roundabout follows an anticlockwise direction.

Before we can input the various scene attributes we need to conduct an initial scene analysis, the results of which are shown in figure 3.11. Part (d) shows the road surface as one continuous region, which in (e) is cut into entry and exit roads, and which themselves are cut into lanes in (f) to which direction of typical traffic flow can be attached. Parts (h) and (i) show the cycle-lane and (b) and (c) the tram-lines. The decomposition describes different continuous regions of space which when put one on top of the other cut space into the smallest sub regions we have called leaf regions, the results of which are shown in figure 3.12. The shape and size of the leaf regions are defined by overlapping the boundaries of the composite regions (themselves defined by the property they express).

By using the composite regions described in section 2.1, we have formed a hierarchical database, an example of which is given in figure 3.13. These composite regions express: (1)
Figure 3.12: The leaf regions in the Bremer Stern map.

Figure 3.13: (1) is the road surface, (2) the entry and exit roads, (3) a turn-right zone, (4) a part of the roundabout, (5) the lanes of the entry road, (6) a give-way zone, (7) a sub-part of the turning zone, (8) the give-way-to zone for 6, (9) contains the leaf regions.

Figure 3.14: Some example give-way regions and their give-way-to zones.
Figure 3.15: Region names and attributes. (a) The names of the leaf regions, which are used in the formation of composite regions. (b) Sets of composite regions selected by attribute (behavioural or physical).

the types of region used by scene objects (e.g., roads, cycle-lane, tram-line); (2) regions of behavioural significance (e.g., give-way, turning); (3) direction information, such as this road leads onto the roundabout; (4) the connectivity of leaf regions. Figure 3.13 depicts a section of the roundabout which has an entry and an exit road (see label 2) which are crossed by a cycle lane and pedestrian crossing. Regions that have a behavioural property include the turn-right zone (labelled 3), the give-way zone (labelled 6) and its associated give-way-to zone (labelled 7 which is the space where a vehicle might be, if for example, it has caused a vehicle at give-way-zone-6 to stop). This give-way (R287) and give-way-to (R288) pair are shown in more detail as part of figure 3.14. Figure 3.15(a) provides some named regions, some of which are used to define the various sets of composite regions given in figure 3.15(b), for example, PEDESTRIAN has \{R83, R84, R89, R90\}; CYCLE has \{R86, R87, R93, R162\}; ROUNDBOUT-SECTOR has \{R157, R482\}; FAST has \{R38, R87, R90, R92\}; CENTRAL has \{R68, R89, R91\}. We can also describe the attributes of the parents of a leaf region, for example, R93 has \{CYCLE, ROUNDBOUT, \}; R90 has \{ROAD-1, ROAD-VEHICLE, PEDESTRIAN, GIVEWAY, INBOUND, FAST, LANE, CARRIAGeway, \}; R157 has \{OUNDBOUT-SECTOR, TURNING, ROAD-1, ROAD-VEHICLE, \}. 
Spatial representation and reasoning play important roles in addressing the surveillance problem. We began by describing the mathematical foundations that have been used to support the region representation used here. These foundations were outlined in chapter 2 section 2.3 with more detail given in appendix B. This foundation has allowed the development of a representation of space that is free from the concerns of vector or raster representation. It can be supported by both, relegating them to implementation issues.

The regions are held in a hierarchical database that enables the separation of spatial extent and structure from the attributes that are attached to a space. The use of a hierarchical decomposition based on enclosure is not new (see chapter 2 section 2.2.7). In addition to describing geographic scene properties the database also holds behavioural information that is determined by what typically takes place in a scene location. These spatially describable behaviours have local spatial invariance, providing an additional source of spatial information. We call accessing this information contextual indexing, because the behavioural information provides a background context that can be used to describe the activity of a scene object.

The important features of this database are the efficient storage of leaf regions, the fusion of topological and metrical data and the hierarchical structure that holds the leaf region relationships for composite regions, neighbours and semantic attributes. The SPATIAL-LAYOUT database plays a supporting role in the runtime system and is organised so that the invariant data is available and can be accessed by simple lookup (e.g., neighbour relationships). The objective was to provide the information required by the runtime system in an efficient way.

The novel properties of the SPATIAL-LAYOUT program include:

- The use of cellular topology to provide a foundation and abstraction that supports different implementations.
- The separation of the representation into spatial extent and spatial attributes for the attachment of behavioural semantics.
- Contextual indexing, which is the provision of spatial information (behavioral and physical) based on the place of an object in the scene. Although obvious, this does not appear to have been addressed before. We have also shown that contextual indexing scales well with scene complexity.

An extension that would be possible in a more ideal vision system concerns the integration of intermediate- and high-level vision. The analogical nature of the spatial representation to what is seen by the official-observer could allow intermediate- and high-level processing to share the ground-plane analogical representation with a common prediction/expectation and analysis to allow contextual indexing and appropriate feedback to the operators that act upon the image-plane data.

The spatial representation presented here provides the spatial framework that will be used extensively in the next chapter.
CHAPTER 4

EVENTS AND BEHAVIOUR

In this chapter we describe how objects can be represented in spatio-temporal terms by the HIVIS-based systems. Our everyday world is situated in space and time, so the problems associated with representing such information are not immediately apparent, in part, because we achieve perceptual understanding of activity without any noticeable effort. This spatio-temporal representation follows on from the description of the environment in chapter 3 where we introduced contextual indexing. Here we develop a discrete model of the dynamics of objects that the perceiver observes in the scene. As described earlier, in chapter 1, we are given data describing these dynamics from intermediate-level visual processing. We use a computational approach that integrates these static and dynamic spatial data to develop a qualitative behavioural description of what is happening in the scene.

This behavioural description can form the basis for further processing that creates a natural language (or conceptual) description of the scene as described in Nagel [171], Neumann [173], Thibadeau [232]. However, the development of full linguistic descriptions is outside the scope of this dissertation. In part, this is because natural language systems tend to take an off-line form, for example the query-based systems of Retz-Schmidt [198] and Neumann [173] (although, as Chapman [34] has demonstrated with instruction use in his SONJA system, this need not be the case). The main reason is that to integrate natural language into the HIVIS-based systems would add more complexity than is necessary to address the surveillance problem described here.

In this chapter we provide two approaches for the computational estimation of space-time conceptual descriptions. The first formulation is simply data driven, using passive vision, to provide a testbed. The second addresses the problem of active vision and requires a complete reformulation in terms of the “here and now”. This reformulation is made in response to a reassessment of the first HIVIS approach.

†In this chapter, sections 1 to 4 are based on the journal paper “An analogical representation of space and time” (Howarth and Buxton [105]) and the conference paper “Analogical representation of spatial events for understanding traffic behaviour” (Howarth and Buxton [106]). Also, sections 6 to 7 are based on the workshop paper “The control of spatial reasoning for high-level vision: a traffic surveillance example” (Howarth [104]).
1 General approach

The perception of object behaviour is a complex task. In general it involves a function
that maps the continuous space of spatial-temporal signals present in the world to the
continuous space of understanding the perceived behaviour. By this we refer to some
ideal understanding of what takes place in the field-of-view. This understanding appears
continuous to us while we are awake, providing a model of understanding to aim at. We
described Newtson [175] and From's [77] work on this in chapter 2. To simplify this
problem we are using (1) a discrete model of spatio-temporal signals in the form of a
stream of time stamped compact-encodings in conjunction with a model of the spatial-
layout of the environment (which is assumed to be static as stated in assumption 3), and
(2) a discrete model of activity in the form of a hierarchy of events and episodes. As
shown in figure 4.1 we do not go straight from the space of spatial-temporal signals to
the space of understanding the perceived behaviour. Instead, we use a number of less
complex functions, that together provide the desired functionality. We assume that there
is a mapping between the perception of an activity and the understanding of that activity.
However, there is not necessarily an equivalence.

There are two HIVIS-based systems developed in this dissertation, called HIVIS-
MONITOR and HIVIS-WATCHER. In this chapter we mainly describe HIVIS-MONITOR,
although towards the end when we address some of the problems raised by this system we
are laying the ground work for HIVIS-WATCHER. As shown in figure 4.2 both HIVIS-
systems use the same input, the stream of compact-encodings. In HIVIS-MONITOR
behaviour perception is viewed as a feature monitoring process. It was developed as
part of the VIEWS project (see Howarth [105, 106]) and is really just a testbed that
was implemented to help identify appropriate spatial representations and the forms of
spatial reasoning likely to be needed in the surveillance problem. HIVIS-MONITOR
uses structural analysis, is "data-driven", and makes use of traditional AI information
processing approaches that we know how to build, and has the objective of supporting
event-reasoning. Event-reasoning concerns the identification of events and their composi-
tion into behavioural descriptions. It is part of the computational theory that supports
the propagation of visual information (in the form of compact-encodings) via spatial
primitives to episodes, and which is illustrated in figure 4.3. Event-reasoning consists of
two processes: "event detection", which selects relevant subsets of these primitives, to
obtain "events"; and "event composition", which incorporates the detected events into an
evolving history from which "episodes" can be identified.

Being a testbed for developing the runtime functionality that accesses SPATIAL-
LAYOUT, led to the selection of a simple approach. However, its use of passive monitoring
has highlighted a number of problems that are likely to be present in any implementa-
tion that employs this data driven approach. The resolution of these led to the development of
HIVIS-WATCHER, which provides a purposive analysis of behaviour. Before describing
HIVIS-WATCHER we discuss HIVIS-MONITOR, beginning with the representation of
the domain-layer compact encodings that is common to both HIVIS-systems.
1.1 COMPACT ENCODINGS

The results from intermediate-level visual processing are a stream of time stamped compact encodings that provide the data about each visible object together with a unique identifier that allows correspondence between successive updates to be maintained. The compact encodings take the form of a sequence of ground-plane outlines of each observed object called pose positions. The "shape" of the pose position denotes both the extent of the object and its frame-of-reference for the object's front, back, left, and right. As shown in figure 4.4 we can model a rigid object using a posebox. This representation does not accommodate all types of objects, for example, rocks, clouds and trees which do not have a specific front to which a frame-of-reference can be attached, and people who are more flexible, having more than one frame-of-reference such as body, head, hand etc.. Even for dynamic objects like people this approach may still be appropriate if, as shown in figure 4.5 a coarse posebox denoting body orientation can be obtained.
1.1.1 Requirements

Given this data, we formulate a number of requirements, such as: describe the evolving path of each scene object as it travels through the scene; describe the interaction between scene objects; and obtain a behavioural description for each relevant object. This will provide the basis for a conceptual description of what is happening in the scene, for example, in the road-traffic domain, "count all cars that turn right" or, more complex, "identify all vehicles that overtake". These requirements differ substantially from those where a spatial representation is used for map learning (see chapter 2 section 2.2.2), navigation [131], robot motion planning (see chapter 2 section 2.2.3). The key differences are that: the path travelled by the objects is (incrementally) provided (see assumption 4), and the environment in which the objects move is known (see assumption 2). The goal is then understanding the movement of the scene objects.

Spatial primitives, called regions, are used for representing both the static ground-plane and the dynamic space occupied by an object’s pose position. The key features required from the static spatial representation described in the previous chapter are:

(1) Convert the ground-plane coordinate data into regions.

(2) Describe the connectivity between these regions.

The objective of the work in this chapter is to reason about the objects in the scene so we also need:

(3) Spatial extent and labelled front, sides and rear to determine occupied regions.

(4) Velocity and orientation to derive spatial behaviour.

When we include multiple moving objects, we also need:

(5) Inter-object orientation and inter-object distance.

These requirements form the basis for our selection of spatio-temporal primitives that take their information from the supplied intermediate-level compact encoding. This interface between the components is due to implementation constraints outlined in chapter 1, and
the use of spatio-temporal primitives is to minimise the effects of this interface allowing the techniques described here to be adapted to more general low- and intermediate-level vision architectures.

1.1.2 Noise

The data supplied in the compact encoding suffers from noise because, although it has been converted to ground-plane coordinates, the data includes the inherent problem that, as something moves further away from the camera (with a fixed geometry), it becomes increasingly more fuzzy and smaller. Remember that the map is a ground-plane view so, ideally, the vehicle should remain the same size (and resolution) all over – clearly this is not the case with a fixed single camera viewing the scene. The effect of this is that positional uncertainty, with respect to the scene objects, increases with depth from the camera. This problem is better dealt with by low- and intermediate-level vision because these stages have more visual data available to them. In comparison, high-level vision can either try to “correct” the dynamic data or incorporate some measurement of the position uncertainty into the static spatial database with respect to the camera position. Neither of these tasks is easy, and, as shown by the previous chapter, SPATIO-LAYOUT does not use spatial uncertainty. However, it is a subject for future work where two approaches appear relevant: (1) the work of Durrant-Whyte [64], who describes geometries designed for reasoning with uncertain data; and (2) “tolerance space” (see appendix B section 6), which provides a way of describing a tolerance for saying how near two objects are, and could be adapted to provide a graduated tolerance linked to the perceptual acuity of the scene. These approaches to the positional uncertainty problem use off-line pre-computation storing the uncertainty measures into the spatial-layout database. This would reduce the effect of uncertainty reasoning on runtime performance, and take advantage of the known scene geometry and geometric information from our fixed camera. Although the problem of positional uncertainty is an important issue we will not address the specifics, except by the use of qualitative values (where possible) to reduce the effect of noise, as described in this chapter.

1.2 Common Primitives

To address the requirements outlined in section 1.1.1 we begin by describing how the compact encodings are converted into primitive properties associated with the observed scene objects. The spatio-temporal primitives are used to identify events, with the values held by the primitives updated when a new time stamped compact encoding arrives. The primitives functionality cannot be altered by task or strategic-layer control, although the control layers can select which primitives are to be used (i.e., they can switch the “sensor” on or off). The primitive properties are derived both directly from the compact encoding and indirectly via SPATIO-LAYOUT. We describe this separation under the headings tracking and spatial context.
orientation is the angle of the tangent, \( t \), to the datum axis in the global coordinate system

- heading uses the vector of motion, \( v \), from the object's centroid, to calculate the vectorial angle (i.e., angular displacement from the datum axis in the global coordinate system)

- when heading \( \approx \) orientation the object is travelling forwards

- when heading \( \approx (\text{orientation} \pm \pi) \) the object is travelling backwards

**Figure 4.9: Heading and orientation.**

1.2.1 **Tracking**

The intermediate-level visual processing should be able to supply values for velocity and acceleration, however, they are not present in the compact encoding supplied here (see figure 1.5 in chapter 1), and are instead replaced by approximations. Figure 4.6 depicts the use of 2D ground-plane centroid values that have been calculated from an object's stream of pose positions. The successive update of these centroids trace out a linked-rod model of the object's path as shown in figure 4.7, with the noise present in the creation of the pose position is also present in the approximations. In addition to the heading supplied by the velocity estimate, we also have the orientation of the object which is part of the pose position data. Figures 4.8 and 4.9 illustrate how the value for heading may differ from orientation. Each arrow attached to one of the dotted lines that connect two poseboxes shows the heading, and the frame (the L shape, denoting the tangent, \( t \), and normal, \( n \), for details see Bruce and Giblin [25, page 30]) attached to the centroid of the pose shows the orientation. The orientation provides a better description of likely forward motion when an object is moving slowly, because of noisy position data.
To overcome the problem of noise in the given data, we can reduce the value range to a less dense qualitative range. The value range of the properties for orientation are reduced from $[0, 2\pi]$ to $[1,9)$ and illustrated in figure 4.10. The value range for the perceived object's speed is converted to the range $[0,8]$, for example in the road traffic domain since most traffic on the roundabout travels at less than 30 mph we convert the range $[0,30]$ mph to $[0,8]$, this range is discretised as shown in figure 4.11. The quality of the data supplied by the pose positions could be improved by the use of a filter. There are a number of different types of filter (we described some in the context of tracking in chapter 2 section 3.4). Bar-Shalom and Fortmann [15] provide a good description of Kalman filters. A Kalman filter can be used to track an object's motion on the 2D plane by representing the "state" of the object at time $t$ with values for: position $x, y$; velocity $\dot{x}, \dot{y}$; and possibly acceleration $\ddot{x}, \ddot{y}$. Typically these values are held in either in one 2D matrix or two 1D matrices. The Kalman filter does not really perform smoothing, which concerns estimating past values of the state on the basis of measurements made up to the present time. Meditch [156] presents an extension to the Kalman filter that does smoothing, however, it is considerably more complex than that for the "pure" Kalman filter. Another approach would be to use median filters (see Tyan [242] for details). We do not filter the object position data in HIVIS because we are using a qualitative approximation, although it is likely to be used in the intermediate-level vision component (see for example Marslin et al. [151] and Koller et al. [124]).

1.2.2 Spatial context

The second source of basic property information is from SPATIAL-LAYOUT, which we access via the position and spatial extent of each object to find out which leaf-regions the object is occupying. We call this intersection test between the leaf regions in SPATIAL-LAYOUT and the object description "occupied-regions" and the implementation of this function uses regions-a-cell-is-in which we detailed in chapter 3 section 3. The occupied-regions test returns information about which part of an object is in what region, in effect, tessellating the object. Since each object has a spatial extent, some more so than others (in the road-traffic domain examples include lorries, trams), it is useful to identify which part of the object entered a region and which part of the object has exited a region. In figure 4.12 we see how the ground-plane leaf regions partition the object shape into the part of the object in regions A and the part in region B. We use this partitioning to say which face of the posebox shown in figure 4.13 is in which region. We could elaborate on this to say what percentage of the posebox is in what region, but the face information gives a sufficient estimate of region occupancy. The result from occupied-regions provides four sets of occupied leaf regions, one for each face of the posebox, and from these sets, we can again access SPATIAL-LAYOUT to obtain the semantic data attached to the composite regions that each leaf region is a part. The result of accessing this semantic data provides a description of the region types that the object occupies. Table 4.1 gives example lists of road traffic and office domain region types and, by using the semantics associated with
the occupied space partial meaning can be attributed to each object's activity by indexing data about the behaviour commonly attributed to that part of the environment.

The spatial context of an event can change its interpretation. For example, consider a sequence of pose updates placed on one part of the ground-plane giving interpretation \( I \) and then moved (translated in \( x, y \) ground-plane coordinates) to another part giving interpretation \( J \). It is unlikely that the new interpretation \( J \) will equal \( I \) because of its new spatial context, even though the actions of the object are the same. The place in which an action occurs contributes to its meaning, that is, it is situated in the environment.
1.2.3 Summary

In this section we have described the spatio-temporal primitive properties that are computed for each object that is given a time-stamped compact encoding. Some of the values, such as velocity, require retaining values, however, the history that needs to be maintained to perform this is of a constant length and fixed overhead.

These primitives are similar to those described by Leutzbach [143] although here our objective is understanding the behaviour of single, binary, and multiple scene objects and not deriving some statistical measure of the flow of these objects through the scene. The primitives described here form an initial set, providing a more meaningful representation of the compact-encoding data.

We have described four functions called orientation, speed, heading, and occupied-Regions. These functions use the following variable types: poseBox which describes an object’s posebox, centroid is a pair of coordinates, numθ is the set of integers \{0, \ldots, 8\} or \mathbb{Z}_8, numθ is the set of integers \{1, \ldots, 8\} as in figure 4.10,\footnote{numθ \ is more properly described by \{1, \ldots, 9\} mapped onto an annulus so that 1 ± 8 = 1.} trackingState holds the state of an object at time \(t\), and fblr \(\alpha\) is a four tuple of type \(\alpha\) having slots for front, back, left and right values that are related to the four faces of an object.

We will not present full details of the four functions, instead we just give their type specifications using a curried notation to describe the symbol types that the functions use.

- The function orientation is of type \(\text{poseBox} \rightarrow \text{num}\theta\).
- To obtain the centroid values we use the function \(\text{poseBox2centroid}\) of type \(\text{poseBox} \rightarrow \text{centroid}\).
- We can define speed in two ways, using two centroid’s from consecutive updates or by using the state of the object at time \(t-1\) and a centroid. There is the option of using a function speed of type \(\text{centroid} \rightarrow \text{centroid} \rightarrow \text{num}\theta\) that represents the linked-rod model or a function speed of type \(\text{trackingState} \rightarrow \text{centroid} \rightarrow \text{num}\theta\) that holds the previous state information in an intermediate form to save recalculation.
- Similarly with the definition of heading, there are two options. We could use a function heading of type \(\text{centroid} \rightarrow \text{centroid} \rightarrow \text{num}\theta\) or a function heading of type \(\text{trackingState} \rightarrow \text{centroid} \rightarrow \text{num}\theta\).
- To obtain the occupied leaf regions we use a function occupiedRegions of type \(\text{fblr AS} \rightarrow \text{fblr LR}\) where the description of the object is in terms of adjacency sets. This adjacency set description is provided by the function \(\text{poseBox2cells}\) of type \(\text{poseBox} \rightarrow \text{fblr AS}\), which uses the access functions described in chapter 3 section 3.7 to obtain details of the application domain region types.

The functions described here form the set of common primitives that are used by the HIVIS-based programs.
2 Events

The primitive properties in themselves do not provide an understanding of perceived behaviour, such descriptions tend to be represented at a more abstract level we will call events. To illustrate what is meant by an event consider the office environment example given in figure 4.14. In the example we see a person get up from a chair, walk over to a door, close it, turn, and walk back to his/her chair. Each segment in this example is a discrete, meaningful action. And the whole sequence can be combined to form one activity closing-the-door. This activity itself takes place inside larger activities, etc. This example also shows the refinement of activity and illustrates some of the levels of description that are possible. To relate these activities to events and episodes we use a temporal representation called “cellwise time”. This representation applies the cellular topology to modelling the temporal axis, allowing us to use topological properties in this representation to describe both continuous values and sharp-changes in values. We introduced this language in chapter 2 section 3.3 and give more details in appendix C section 1.1 where we describe how this language helps to make clear the effects of using a stream of frame-updates. In cellwise time, activities are made up of one or more time-cells and state-changes occur between two time-cells. We identified previous work in chapter 2 sections 3.2.1 and 3.2.2 on how the behaviour of others is perceived as discrete units. To identify the closing-the-door episode we need to identify the two significant state-changes that are the events between “typing” and “get-up” and between “walk-back” and “typing”. These events in themselves do not describe the closing-the-door episode, they only bound the activity. These two events are part of any activity that involves getting up and sitting down again, such as answering the telephone or opening the window, and themselves do not describe closing-the-door. Identifying episodes like closing-the-door is difficult and in this section we just consider the less complex problem of how to describe and detect events.

In chapter 2 section 3.2.2 we introduced Thibadeau’s work [232] that is based on Newtson’s studies [175] of how actions are perceived. Similar to our use of compact encodings, Thibadeau makes the simplifying assumption that the given perceptual input is a sequence of frames, $f$, where each frame, $f_t$, is an interpreted image for a moment $t$ in time. Thibadeau describes two classes of change in the stream of successive frames: a simple state difference or first order change, denoted $c^1$; and a change in a change or second order (derivative) change, denoted $c^2$. The number of moments contributing to these changes is not specified. Thibadeau describes a generative approach that he claims would produce the “superset of all actions in an activity” and also points out that the number of generated actions would be large even with the simplification of just using the differences between two successive states. This problem is worse for Thibadeau who is dealing more directly with real world imagery, which produces an enormous number of change descriptions over a short period of time. In contrast, we are using compact encodings which simplifies our problem. We will make use of Thibadeau and Newtson's
classes of change, together with their action postulate that defines "points-of-action" as being drawn from the collection of second-order changes, $c^2$.

**Definition 4.1** An event is a significant change to a first-order primitive properties.

This definition conforms to the point-of-action definition given by Newtson [175]. We can also define activities that take place between events as:

**Definition 4.2** A steady activity is a significant continuous value of a first-order primitive properties.

In this definition of activity we also include durations where motion is zero, for example, "sitting". The problem of "recognising an event" becomes that of recognising when a first-order primitive property changes its value. This requires having a memory that holds the previous property value, and also knowing what constitutes a state-change for this property.

For some property $p$ of an object $o$ we can denote a first-order change by the formula $\text{TRUE}(f_{t-1}, f_t, c^1(p(o, v_{t-1}^p), p(o, v_{t}^p)))$ that describes how the value $v_{t-1}^p$ changes to value $v_{t}^p$, where the two values are drawn from the set of allowed values for property $p$. We can also use this representation to describe second order changes in terms of $c^2$ changes with the formula $\text{TRUE}(f_{t-1}, f_t, c^2(c^1(o, v_{t-1}^1), c^1(o, v_{t}^1)))$. The detection of events assumes that state-changes can be used to describe all the necessary features for providing a conceptual description.

We use primitive properties to describe three types of events that are classed according to the number of objects involved, and are called “single”, “binary” and “multiple” object events. Single object behaviour concerns those properties personal to the object such as its
<table>
<thead>
<tr>
<th>properties</th>
<th>c^1</th>
<th>activities</th>
<th>c^2</th>
<th>events</th>
<th>cellwise-time of update</th>
</tr>
</thead>
<tbody>
<tr>
<td>update for object</td>
<td>continuous update</td>
<td>u ∈ compact encoding</td>
<td>change in presence of update</td>
<td>enter-scene</td>
<td>( (-u) \mathbb{I}(u) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>exit-scene</td>
<td>( (u) \mathbb{I}(-u) )</td>
</tr>
<tr>
<td>pose position</td>
<td>continuous spatial context</td>
<td>sc ∈ [f1br, L1r]</td>
<td>change in spatial context</td>
<td>enter-region</td>
<td>( \exists fr \in (front sc) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( (-fr) \mathbb{I}(fr) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( \exists br \in (back sc) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( (br) \mathbb{I}(-br) )</td>
</tr>
<tr>
<td>centroid position</td>
<td>velocity</td>
<td>v ∈ num^β</td>
<td>acceleration</td>
<td>faster</td>
<td>( (v) \mathbb{I}(v + \delta) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>slower</td>
<td>( (v) \mathbb{I}(v - \delta) )</td>
</tr>
<tr>
<td>displacement</td>
<td>motion ∈ {moving, stationary}</td>
<td></td>
<td>stop</td>
<td>( \mathbb{I} )</td>
<td>( \mathbb{I} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>start</td>
<td>( \mathbb{I} )</td>
<td>( \mathbb{I} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( \mathbb{I}(moving) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( \mathbb{I}(stationary) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( \mathbb{I}(moving) )</td>
</tr>
<tr>
<td>heading</td>
<td>θ ∈ numθ</td>
<td>change heading</td>
<td>turn-left</td>
<td>( (θ) \mathbb{I}(θ + \delta) )</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>turn-right</td>
<td>( (θ) \mathbb{I}(θ - \delta) )</td>
<td></td>
</tr>
<tr>
<td>pose orientation</td>
<td>deviation</td>
<td>direction ∈ {forwards, backwards}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Single object properties, activities, and events. The cellwise-time notation describes a test for the presence of a value that uses \( \mathbb{I} \) to denote “not-present”.

speed, heading, orientation and the place it is located. Binary object behaviour concerns more the relationships between pairs of objects such as their spatial arrangement. Multiple object behaviour expresses the more general form of the binary case. These different perspectives are needed to fully understand the perceived behaviour.

### 2.1 Single Object

Single object events draw upon the primitive properties based on position, such as changes in speed, heading, orientation and region occupancy. The detection of events such as stop, start, faster and slower are expressed in table 4.2. The primitives given in this table are necessary and complete for describing the restricted set of single-object events used in the HIVIS-MONITOR examples (see section 4). However, this restricted set is not rich enough to provide a complete description of object behaviour needed in the surveillance problem, but does provide a useful starting point from which to build a more complete model.

Figure 4.15 shows how there may be a number of single-object events that access the same first-order primitive property, with each identifying different significant changes. The qualitative value ranges make this easier, by allowing more concise descriptions to be made about the continuous values, state-changes and connectivity of a signal. In addition to these values for orientation, heading and speed, there are spatio-temporal representations associated with the history the object forms and the regions boundaries it crosses and the regions it occupies.
2.1.1 Conduits

We define a representation called a “conduit” to describe the space swept out in time by an object’s path, which is constructed from pairing consecutive poseboxes as illustrated in figure 4.6. The path of the object is represented by “conduit-construction”. The conduit is a simplified form of the extrusion described in Cameron [31], since it does not represent a complete 3D extrusion but this could be done using 3D cell solids. It also corresponds to a simplified form of Hayes’ [95] “histories”, covering a much more restricted set of physical properties. The conduit can be used to approximate, more accurately, the time at which a region was exited or entered. To do this, we extrapolate the space-time description between updates. The path formed will depend upon the curve or line used to connect the points and whether knowledge about the regions is used (for example, in the road-traffic domain cars usually stay on the road). This method of small scale path completion can also be applied to reasoning about missing updates due to error or occlusion. Conduits capture in spatio-temporal form the past history of an object’s passage through the scene, providing an unusual representation of continuity of travel. We have tried interpreting these spatio-temporal volumes (for example see appendix C section 2.2.2), but this was seen as being a dead end because of the complexity encountered when extending this approach to describe binary and multiple object interactions. Although it does not extend well, conduits do provide a useful framework for illustrating region transition and occupancy.

2.1.2 Region boundary transitions

An example of recognising events is given by the description of how an object travels over a ground-plane, where the ground-plane is composed of semantically meaningful regions, as described in section 1.2.2. In terms of definition 4.1 a region transition is a state-change that describes the action of crossing a region boundary either entering or exiting a region. This gives us an abrupt change in properties, with an associated region boundary between two cells. A state change is called an event to denote its “instantaneous” change.
(e.g., the front edge of an object passing over a region boundary, signifying a boundary crossing event). Figure 4.16 illustrates a region transition where the object enters the new region and exits the old region. However, not all region transitions involve the object fully occupying the new region, figure 4.17 illustrates how an object may only partially occupy a neighbouring region. The charts under each of these illustrations have the time axis along the bottom with the numbers corresponding to the frame segments above. The bars in the chart denote the temporal interval during which the object has a face is in a particular region, these bars show that using the faces of the object is sufficient for distinguishing between these two cases. Figure 4.18 shows a region crossing of the kind given in figure 4.16, illustrating the dual relationship between entries and exits. These can be represented by a list of region-names that are altered to reflect the value changes of the primitive properties enter-region and exit-region in table 4.2.

Figure 4.19 illustrates the continuous property of region occupancy while the object is in a region. At the top left of the figure is the ground-plane with an object travelling over it, on the right is a sequence of frames, bottom left is the conduit, and in the middle is the duration of region occupancy in space time. In the figure some of the frames describe events: 2 is the earliest entry, 3 is the latest entry, 5 is the earliest exit, and 6 is the latest exit. The numbers denote time cells. From this we can say \text{TRUE}(2,6,\text{inregion(OBJECT24,REGION17)}) the time that the object is in the region. The 3D volume provides a region of spacetime representing how long the object is in region R157.

2.1.3 Region occupancy

In terms of definition 4.2 a region occupancy is a steady activity that describes the action of travelling inside a region, which is seen as a topologically steady state. Although we may cross the boundaries of the child regions and cells that make up this region (or a set of continuous regions that all share the same property value for this property), all the space travelled through has the same "value" (e.g., all the space (i.e., subregions and cells) that make up a region of type \text{central-island} have the same property value), so there is no value change. Travelling about inside a region is similar to having an allowable value range (or interval) in which an analogue meter reports valid readings as long as the measures are always inside the valid range of values.

2.1.4 Discussion

Single object representation and reasoning play a central part in HIVIS-MONITOR, being used primarily to illustrate how the spatial information contained in SPATIAL-LAYOUT can be used to describe its behaviour. The main single object events are itemised in table 4.2. The attention to region related detail has caused the time of entry and exit to be seen as being important. This priority led to the development of conduits and small scale path completion which are used to fill in the information missing from the discrete
Figure 4.16: Region entry/exit.

Figure 4.17: Partial occupancy

Figure 4.18: Region transition involves entering a new region and exiting an old region.

Figure 4.19: A conduit created from a sequence of frames.
Figure 4.20: A non-exhaustive set of labelled binary spatial arrangements.

2.2 Binary Object

There are a number of binary spatial arrangements that we would like to describe such as those shown in figure 4.20. However, they are difficult to define in an absolute coordinate system with the objective of identifying when one arrangement changes into another. In a global perspective it is easier to deal with centroid representations of the scene objects or shapes that are orthogonal to the coordinate axes. Reasoning about other orientations is more complex. We call this the spatial arrangements problem. We do not address this problem here instead, to reduce the need for performing this form of reasoning we use the regions in SPATIAL-LAYOUT to identify when two objects are in the same region.

This approach addresses the problem of whether two objects are in the same region, where a region could be a leaf or composite (e.g., in traffic-domain, we could have a road or section-of-road). The approach we describe here uses topological focusing and contextual indexing. The region tessellation is used to determine when the two cars are near to each other. In this model, near is defined by

\[
\begin{align*}
\text{TRUE}(t_1, t_1, \text{inregion}(x_1, r)) & \land \text{TRUE}(t_1, t_1, \text{inregion}(x_2, r)) \\
\Rightarrow \text{TRUE}(t_1, t_1, \text{near}(x_1, x_2))
\end{align*}
\]  

(4.1)
so we have a statement of the nearness of the objects \( x \) and \( y \), being in region \( r \) at some time-cell\(^2\) \( t_1 \), which gives us a description in space and time. This simple topological definition is data driven; the calculation of near is only made when the region formulas are created, and are monotonic for the given time point. The cost of managing the \textit{inregion} calculations is a constant overhead on top of the cost of processing each object update, making the calculation of near order \( O(n) \), plus the cost of performing the intersection test. This is particularly useful in the situation where we are dealing with more than two objects at any one time. Compare this with a simple geometric technique in which the distances between all cars in the scene are calculated, giving an \( O(n^2) \) time complexity. This assumes that the calculation does not use any ground-plane tessellation information, and is the worst case value.

We can generalise this by making use of the time-intervals in our implementation. The intervals from two objects form bounded spacetime descriptions of when the two objects are near each other. These intervals can be mapped onto a time line describing the behaviour of the two objects. For example, two objects being near each other which could prove initial evidence for one object following the other. We will denote the concept of two objects being in the same region as for some interval \( \{i, j\} \)

\[
\begin{align*}
\text{TRUE}(i, j, \text{inregion}(x_1, r)) & \land \text{TRUE}(i, j, \text{inregion}(x_2, r)) \land \\
\text{\( i \) is max}(i, i_2) & \land \text{\( j \) is min}(j, j_2) \land i < j \\
\Rightarrow & \text{TRUE}(i, j, \text{insameregion}(r, x_1, x_2)) \\
\end{align*}
\]  

(4.2)

In formula 4.2 we have also explicitly stated the interval manipulation, which was simplified in formula 4.1. In addition to considering this in terms of symbols, we can also describe two objects being in the same region in terms of the space time volume that each object is in a region. An example of how a space time volume can be formed is given in figure 4.19, with figure 4.21 showing how using the infix operator \( \oplus \) can be used. The \( \oplus \) operator has similar functionality to that expressed in equation 4.2. We can extend the binary object case to the more general multiple object case.

2.3 Multiple Object

In this more general case we identify the time intervals during which more than one object is occupying the same region. When we have more than two objects in the same region we use a more general definition of formula 4.2

\[
\begin{align*}
\text{TRUE}(i, j, \text{inregion}(x_1, r)) & \land \cdots \land \text{TRUE}(i, j, \text{inregion}(x_n, r)) \land \\
\text{\( i \) is max}(i_1, \ldots, i_n) & \land \text{\( j \) is min}(j_1, \ldots, j_n) \land i < j \\
\Rightarrow & \text{TRUE}(i, j, \text{insameregion}(r, x_1, \ldots, x_n)) \\
\end{align*}
\]  

(4.3)

\( ^2\)The implementation uses time intervals, so in formula 4.1 we really have \( i = [i, i] \). The cellwise-time language does not support zero duration intervals because an interval must contain at least two distinguishing cells (see definition C.5 in appendix C).
The end result is a set of \textit{insameregion} assertions which are temporally discrete for the same region. Achieving this result requires some extra manipulation between current intervals for the same region which provides the time line description be used to reason about the multiple–objects near behaviour.

The implementation uses interval-trees details of which are given in appendix C section 2.1. This method for calculating multiple object region occupancy has \( O(n + k) \) time complexity, where \( n \) is the number of objects in the scene at any one time, and \( k \) is the number of “objects in the same region”. The multiple object events derived from this interval structure are related to the objects that are in the same region and stored in the Temporal History Builder (see section 3.3.2). Figure 4.22 shows an example of three people walking along a corridor separated into regions by fire-doors. The events generated from the graph identify when the current list of occupying objects changes. In the above example we would have an event at \( t_2 \) and \( t_4 \).

As a measurement of “nearness” it is dependent upon the size of the region, which may be just what is required in situations such as identifying which objects are in the same room or stretch of road. A more consistent measure of proximity is provided by the aspect nearest-object-to-me.

2.4 Summary

We have described the first stage of event-reasoning, providing a mapping from primitive properties to events, and in the next section we will describe a compositional model that uses events to describe episodes. In the data driven HIVIS-MONITOR system the set of operators, \( \Omega \), provides the event detection stage illustrated in figure 4.3. For each frame of data each operator \( \Omega_1 \) is applied to each scene object.

\textbf{Definition 4.3} An operator \( \Omega_1 \) is a function of the form
\[ \text{op}_1 : \text{args} \rightarrow \text{results} \]

and is used to obtain the \( i^{th} \) primitive property value.

Since property values are associated with objects we need to supply a given object from the set of objects. In general each operator takes a set of args and provides a set of results and/or side-effects to some global storage. There is an operator for each event listed in table 4.2 plus the multiple-object events when necessary.
3 Episodes

In this section we address the second stage of event-reasoning. We investigate how to develop a framework within which conceptual descriptions of scene behaviour can be constructed from the events and activities described in section 2. In this computational framework the results from composing events into episodes are added to an evolving database. This database is intended to be used by an off-line query system so that a user can find out what happened in the scene. This is an ambitious objective which is difficult (if not impossible) to fulfill because we do not know what questions may be asked. We do not fully address this objective, instead the framework we develop here tries to extract all occurrences of a predefined set of episodes which can be linked together, providing an impoverished set of conceptual descriptions. Failing to obtain all conceptual descriptions is not a major setback because as mentioned earlier we are not addressing issues related to natural language or query based systems. Our interest lies in how the SPATIAL-LAYOUT can be used to form these conceptual descriptions. Within the more tractable objective we can identify two problems episode completeness which concerns how to detect all instances of a given set of episodes, and event composition which concerns how to describe episodes in terms of events. We do not address episode completeness in the prototype HIVIS-systems developed here, we just demonstrate that episodes can be detected using the computational framework. However, we do discuss event composition.

3.1 Event Composition

This is the third stage illustrated in figure 4.3. The input to the event composition problem is a stream of events that are related to scene objects. The objective is to identify the various episodes that are present in this input stream of event assertions. In section 2 we identified a set of relevant events, and in some respects it is difficult to see why identifying episodes from a stream of events is difficult because an episode is just a pattern of events. We do not need a perfect solution to illustrate how SPATIAL-LAYOUT can be used, but even to obtain a partial solution we need to identify the problem in more detail. The component missing from the description of events and activities is how they are related on the temporal axis. When we compose events we are making use of two general temporal orderings, sequential ordering SEQ which concerns how events compose when one occurs directly after the other and concurrent ordering SIM concerns when one event happens at the same time as another. The cellwise-time language provides a discrete model of time in terms of cells, with the principal interpretation being for supplying semantics to pairs of sequential activities (for more details see appendix C section 1.3).

To get an idea of the size of defining this compositional model for one form of interpretation we will use a set of events, EV, and a set of partial episodes, PE, and express the

---

3 SEQ and SIM correspond to sequential execution of command lists (denoted ";") and parallel commands (denoted "||") in Hoare's [99] Communicating Sequential Processes language. This relationship with CSP was not developed because the intention here was the development of a simple computational model.
Figure 4.23: An illustration of a subset of the range and domain of $P$, where $z$-axis = $P_x$-axis, $y$-axis.

Figure 4.24: Hierarchy of behaviours.

event composition as a function $\text{comp}$ of type $[EV] \rightarrow PE$. We could define $\text{comp}$ as a lookup table, LT, by itemising the various subsets of EV that contain two, three and four elements and describe the meaning of the permutations of these elements. Each permutation is an element in LT such that it is either a nameable episode or an invalid sequence. The modulus of LT provides the number of all sequential event combinations less than length five. To find out how large this set LT of permutations is, let us define a function $P$ which is described by $P(n)(l) = \sum_{i=2}^{l} P_i^n$ where $n = |EV|$ and $l$ is the maximum length of an episode. If $n = 6$ and $l = 4$ then $P = 510$, making this form of completeness unrealistic even for small numbers of events. Figure 4.23 shows that even if we limit the length of episodes to two events completeness becomes slightly more feasible. However, the reason for being interested in episodes that contain more than two events is that most episodes that we want to recognise are like the closing-the-door episode (see figure 4.14) in that they have two events that bound the episode and a number of lesser events and activities that take place in some sequence. Fortunately completeness may not be needed, describing the combinations necessary for detecting the conceptual descriptions that are currently required is enough although it does appear to be a more ad hoc approach.

3.2 Example episodes

Much of the information episodes capture is application and context dependent. For example, in the office domain we have episodes like closing-the-door, working-in-the-office, opening-the-window, answering-the-phone. As shown in figure 4.24 the road traffic domain has its own set of behaviours that in turn specify possible object actions. The structure present in this application domain enables us to specify a set of

---

4The number of permutations of $r$ objects taken from $n$ different objects is $P^n_r = \frac{n!}{(n-r)!}$. 
typical behaviours that covers most of the perceived activity. In the road domain this set of behaviours includes:

- region-crossing
- entering-onto-a-roundabout
- following
- queue
- travelling-through-a-roundabout
- take-next-exit (turning)
- overtaking
- give way
- waiting
- cross

We will describe a few of these in more detail because they are used in section 4 and also in chapter 5.

3.2.1 REGION CROSSING

A region crossing is composed from two region boundary transitions (see section 2.1.2) and one region occupancy (see section 2.1.3). An everyday example that captures these notions is walking along a corridor cut into sections by fire doors. You open the first pair of fire-doors, walk along the section of corridor to the next fire doors, open these, so exiting that section, having accomplished the traversal of the connecting corridor.

As an object travels over the ground-plane, it crosses the hierarchical layered regions, generating region entry and exit state changes, without affecting the structure of the ground-plane space (see chapter 3 section 2.2). A region crossing describes an episode, which includes the accomplishment of having travelled through a region, with its associated entry and exit state-changes (e.g., it is composed of at least two significant boundary crossing events). Crossing a region has a number of qualitative topological states as shown in figure 4.25 although the essential property can be more simply represented by a boolean signal as shown in figure 4.26.

3.2.2 ENTERING THE ROUNDABOUT

Entering-onto-a-roundabout consists of two properties: continuously on road surface, and transition from entry road to roundabout. See figure 4.27 which illustrates the transition and also that we can form the expectation that having entered the roundabout,
at some future time, the object will leave the roundabout. The salient features of the
behaviour are captured by the following description of valid roundabout entry:

$$\forall t_1 < t_2 \quad \text{TRUE}(t_1, t_2, \text{ROUNDABOUT-ENTRY}(x)) \Rightarrow$$
$$\exists t_1 < t_3 < t_4 < t_2 \quad \text{TRUE}(t_1, t_2, \text{ON-ROAD-SURFACE}(x)) \land \text{TRUE}(t_1, t_4, \text{ON-ENTRY-ROAD}(x)) \land$$
$$\text{TRUE}(t_3, t_2, \text{ON-ROUNDABOUT}(x))$$

This description of entering onto a roundabout includes at least three events that can be
asserted over time as the episode unfolds. Figure 4.28 gives a sub-lattice from SPATIAL-
LAYOUT which shows the valid sets of composite regions, particularly that ENTRY-ROAD
and ROUNDABOUT are discrete, while ROAD-SURFACE is common to both of them. Although
this lattice does capture the notion of changing region occupancy if we modelled objects
using centroids to represent spatial extent, the lattice is not correct for the posebox model
we are using here. This is because an object’s posebox can stretch across a boundary
between two regions. We will use a short example to illustrate the three events associated
with ROUNDABOUT-ENTRY. To begin with the object OBJ1 will not be in any regions because
it is not in the field-of-view, and we use the $\perp$ symbol to denote this.

- When we receive the first frame of compact encoding describing OBJ1’s posebox position
we can assert that the object enters the field-of-view and is on the road surface and
entry road.

$$\text{TRUE}(u_1, u_2, \text{CHANGE}((\text{REGION-TYPE(OBJ1, L)}, \text{REGION-TYPE(OBJ1, ROAD-SURFACE))))}$$

$$\text{TRUE}(u_1, u_2, \text{CHANGE}((\text{REGION-TYPE(OBJ1, L)}, \text{REGION-TYPE(OBJ1, ENTRY-ROAD))))}$$

- Then at some later time OBJ1 leaves the entry-road and enters the roundabout.

$$\text{TRUE}(u_3, u_4, \text{CHANGE}((\text{REGION-TYPE(OBJ1, ENTRY-ROAD)}, \text{REGION-TYPE(OBJ1, ROUNDABOUT))}))$$

- Then OBJ1 leaves the field-of-view

$$\text{TRUE}(u_5, u_6, \text{CHANGE}((\text{REGION-TYPE(OBJ1, ROUNDABOUT)}, \text{REGION-TYPE(OBJ1, L))))}$$

$$\text{TRUE}(u_5, u_6, \text{CHANGE}((\text{REGION-TYPE(OBJ1, ROAD-SURFACE)}, \text{REGION-TYPE(OBJ1, L))))}$$

This just describes REGION-TYPE changes. Figure 4.29 illustrates other object properties
such as orientation and speed.

\subsection*{3.2.3 Turn-right}

We can describe a valid turn right event as follows:

$$\forall t_1 < t_2 \quad \text{TRUE}(t_1, t_2, \text{TURN-RIGHT}(x)) \Rightarrow$$
$$\exists t_1 < t_3 < t_4 < t_2 \quad \text{TRUE}(t_1, t_2, \text{IN-TURN-RIGHT-REGION}(x)) \land$$
$$\text{TRUE}(t_3, t_4, (\text{ORIENTATION-CHANGE}(x, \theta) \land (\theta < -10)))$$
Figure 4.27: Script model of travelling-through-a-roundabout.

Figure 4.28: The valid region-type combinations for the types ROAD-SURFACE, ROUNDABOUT and ENTRY-ROAD held in SPATIAL-LAYOUT.

Figure 4.29: Entering onto a roundabout.
This decomposes the problem into the detection of features that can be more easily determined. The turning example says that the vehicle $x$ is turning at time $t$ if it is in a turning region and has an orientation change of greater than $10^\circ$ in the clockwise direction. This episode provides a subcomponent of a more detailed version of entering-onto-a-roundabout illustrated in figure 4.29.

$$\forall t_1 < t_9$$
$$\text{TRUE}(t_1, t_9, \text{ENTERING-ONTO-A-ROUNDABOUT}(x)) \Rightarrow$$
$$\exists t_1 < t_2 < t_3 < t_4 < t_5 < t_6 < t_7 < t_8 < t_9$$
$$\text{TRUE}(t_1, t_9, \text{ON-ROAD-SURFACE}(x)) \land \text{TRUE}(t_1, t_9, \text{ON-ENTRY-ROAD}(x)) \land$$
$$\text{TRUE}(t_2, t_8, \text{IN-TURN-RIGHT-REGION}(x)) \land \text{TRUE}(t_3, t_4, \text{STATIONARY}(x)) \land$$
$$\text{TRUE}(t_4, t_8, \text{ON-ROUNDABOUT}(x)) \land$$
$$\text{TRUE}(t_5, t_7, (\text{ORIENTATION-CHANGE}(x, \theta) \land (\theta < -10)))$$

These temporal intervals are shown in figure 4.30.

### 3.2.4 Give way

If a vehicle is stationary for some time in a region where give-way behaviour is expected, then we would postulate that the vehicle is giving way to another vehicle. We also know which are the likely regions where this other vehicle could be (i.e., the give-way-to regions described in figure 3.14 of chapter 3), allowing the system to focus its attention on these regions first. We can describe the basic behaviour as:

$$\forall t_1 < t_2$$
$$\text{TRUE}(t_1, t_2, \text{GIVEWAY}(x, y)) \Rightarrow$$
$$\text{TRUE}(t_1, t_2, \text{WAITING}(x)) \land \text{TRUE}(t_1, t_2, \text{IN-GIVEWAY-REGION}(x, r)) \land$$
$$\text{TRUE}(t_1, t_2, \text{HAS-GIVEWAY-SET}(r, \text{ge})) \land \text{TRUE}(t_1, t_2, \text{IN-GIVEWAY-TO-ZONE}(y, \text{ge}))$$

This description does not fully capture how one vehicle is waiting until the vehicle/s that have the right-of-way have passed and that there is then space in the flow of traffic for the waiting vehicle to pull-out onto the roundabout. The test for giveaway-regions and objects in the giveaway-to-zone is really just an initial condition for identifying the participants in
the episode to the official-observer. However, the behavioural description does illustrate how the give-way region data held in SPATIAL-LAYOUT can be used.

3.2.5 Queueing

Queueing is an example of multiple object behaviour. Here x is a set of objects.

\[ \forall t_1 < t_2 \]
\[ \text{TRUE}(t_1, t_2, \text{QUEUE}(x)) \Rightarrow \]
\[ \forall x_i \in x \quad \text{TRUE}(t_1, t_2, (\text{STATIONARY}(x_i) \lor \text{MOVING-SLOWLY}(x_i))) \]
\[ \exists x_j \in x \quad \text{TRUE}(t_1, t_2, (\text{INSAMEREGION}(x_i, x_j) \land (x_i \neq x_j))) \]

This could be combined with give-way to include the object (maybe a tram) that has caused the queue to form. This description does not really capture the dynamics of queue formation which would be better described using terms like: leave-queue, join-queue.

3.3 Implementation Details

Having provided some examples, here we describe an architecture, for HIVIS-MONITOR that supports the computational model we have outlined. The purpose of HIVIS-MONITOR is to act as a testbed that allows us to determine if SPATIAL-LAYOUT supports the surveillance problem. The functions that access SPATIAL-LAYOUT are described in chapter 3 section 3.7 and are used by the primitives described in section 1.2. The formation of episodes provides a purpose for using these primitives. HIVIS-MONITOR is data-driven and follows the script-based approach, constructing an interpretation of object behaviour in an evolving database that holds entries for the history of each individual object and the interactions between them. This bottom-up approach reflects the flow of data from image to conceptual-description. The main inference-layer mechanism in HIVIS-MONITOR is the database that holds events and episodes. Also to use this database we need functions that can update its contents.

3.3.1 Framework

In general the approach for maintaining a history of the behaviour that takes place in the scene is to note down the event primitives that have been detected and then use an ongoing interpretation process to see how these events fit together. The input given to the database consists of the events and activities associated with a particular property as described in section 2. In addition to the functions that compute these values there are further functions that update the temporal structure by beginning, extending or ending the continuity of the value/signal for each property. To identify an episode we use a filter that extracts the necessary property values from the available properties. To do this we use a script-like language that has the form:

episode1 = activityZ(activityY(activityX
episode2 = activityP(activityQ
where an event is described by the boundary between its two activities and denoted by \( \mathcal{I} \). As we incrementally update the temporal axis, adding new data from each frame update as it arrives, the properties of each object are updated providing a temporally moving present, something that as van Benthem [244, page 6] points out a logical study of time does not pretend to cover.

In AI the world is often represented in terms of world-states that describe the world at a moment in time rather like an image frame in our stream of compact encodings. Often associated with this world-state representation is the frame problem (see McCarthy and Hayes [154]) which stems from the need to represent those aspects of the world that remain invariant under certain world-state changes. In a first-order predicate calculus representation of such world, it would be necessary to explicitly represent all of the invariants under all world-state changes. These invariants are called the “frame axioms” for the world being modelled. For example, we would need to say that when an object changes position it does not change colour, and that when one object sits down or stops the other objects do not do the same. The problem is more complex because there are situations where these frame axioms are false, for example, in a paint shop on an assembly line or at a party. These exceptions tend not to be part of the everyday world, so it might be possible to exclude them and just list the typical and normal frame axioms. Unusual cases then become “breaches of the norm” (see chapter 2 section 4.2). However, if we do this it removes part of the justification for having the frame axioms in the first place.

In HIVIS-MONITOR we do not explicitly use frame axioms for reasoning about the world because we do not use inference to determine whether something is true at a given time. Instead we are concerned with correlating new data with what is currently in the database. Each update increments the temporal axis, and the values from the update occupy one cell in the temporal axis. The update may contain details of a number of properties \( \{p_1, \ldots, p_n\} \) and it depends upon how the new and old values compare as to whether the property value is continuous or has a state-change. During our updating of the various properties we assume that unless we are given a new value for a property, the property is not extended to the current time cell. For example, the object may no longer be visible because it has gone out of the perceiver’s field-of-view. We gave an illustration of the update process in the charts of figures 4.16 and 4.17 that are built out of region transitions. The history mechanism is monotonic, it remembers everything. This is because we do not know when some event will not be needed to compose an episode or if an episode will be needed to compose some more abstract episode description.

3.3.2 Event storage

To store the events and incrementally update the durations of activities we use a mechanism called the “Temporal History Builder” (THB). The THB has some similarities to a simple TMM (see chapter 2 section 4.3.1 and Dean and McDermott [56]). The key difference is that the THB is not able to reason about time in a global way. McDermott [155] calls taking this global viewpoint “looking from the side”, in which the past, present and future
are all laid out and accessible to the reasoning mechanism. The THB cannot perform predictions, which may be seen as a major disadvantage, since producing predictions is often considered to be a useful part of temporal reasoning (see chapter 2 section 4.3.1 and Shoham [214]). The reason we do not include predictions is that they can be a time consuming and unending process. If we were to add this functionality to THB we would also need to address the frame problem.

The THB uses a simple temporal reasoning mechanism that was developed to reason about region crossings. It contains all the necessary functionality for describing the various stages of a region crossing as an episode with a clearly defined beginning “enter-the-region”, an activity “travel-through-region” and an end “exit-the-region”. The temporal reasoning mechanism implements the framework described in section 3.3.1 providing incremental construction of interval. This approach solves the event composition problem by describing an episode in terms of an interval that represents some relevant continuous property. The THB uses a dynamic interval tree (which is described in appendix C section 2.1) to store events and extend activities, with the properties obtained from the new frame of data added to the interval tree for each frame update. An episode can only be used for further reasoning once it has been recognised, i.e., that the final event that completes the episode’s temporal interval has been identified. This means that all reasoning involving episodes is off-line, with episodes only being identified after they happen.

The nodes in the THB are indexed by frameno and objectname, and at these nodes we store a tuple of the form (coords,orientation,centroid,flbr [AS],sbound,flbr [LR]) which we call ODT for “object description tuple”. The coords description of the object is just for the purpose of display. The properties orientation, centroid and flbr [LR] provide input to the calculation of activities and events we describe below. To manipulate ODT’s the THB includes the following functions:

- for converting the input to ODTs we use the function pose2ODT of type poseBox → ODT,
- for updating the THB we use the function putHistory of type frameno → objectname → ODT → THB → THB, and
- for extracting a tuple for a given time-cell and object we use the function getHistory of type frameno → objectname → THB → ODT.

After updating the THB by adding a new node for each object in the frame description we run the algorithm described in figure 4.31. The update THB algorithm first runs a collection of $C^1$ functions to identify any $C^1$ changes between the new nodes and their respective nodes from the previous update. The purpose for this is to identify whether the properties associated with the new node can be linked to the properties for the previous node. After these have been computed we then run a collection of $C^2$ functions to do the same thing for $C^2$ changes between the identified $C^1$ values.

The $C^1$ functions all take the current and previous nodes for an object and extend or create a new interval for their property. This is dependent on whether the value is the same or different. Intervals are described by an “interval description tuple” IDT of the form (value, property, objectname) where property is the $c^1$ or $c^2$ function name. The
for each \( f_t \in F \)
for each objectname \( \in f_t \)
add new OTT for the compact encoding of objectname
for each \( g \in \) the set of \( C^1 \) functions 
and the corresponding \( h \in \) the set of \( C^2 \) functions
run \( g \) on objectname
run \( h \) on objectname

Figure 4.31: The update THB algorithm.

temporal extent of the interval is denoted by the pair of numbers \([t_a, t_b]\) that describe its first and last time-cells in the interval tree. The end cell is extended when the value of the IDT is the same. A temporal boundary is created when the value is different. This boundary is formed by completing the old IDT, and starting a new one.

In HIVIS-MONITOR the \( C^1 \) functions are heading, and speed (which both use the centroid value), and deviation (which uses the orientation value), and spatial-context (which uses the flbr [LR] value). There are also four \( C^2 \) functions that detect changes in the four \( C^1 \) function results. These two stages can be combined as described in the algorithm given in figure 4.31.

The update THB algorithm can be extended to include functions that are run when a particular \( C^2 \) change is identified. An example of this is using the exit-region change of a particular region to select an object’s region-crossing episode. Once we have identified such an episode in the form of an IDT that described the region, the object and the temporal interval during which the object was crossing the region, we can use it to find out if any other objects were in the same region. We described this form of reasoning with episodes in section 2.3 (also see figure 4.21 and equation 4.3).

The extension to the THB interprets the insameregion test as being an intersection of intervals. This approach has a low time cost since the interval tree in the THB is designed to perform this type of interval computation. The property of insameregion is related to the regions not the individual objects, because the boundaries on the time-line represent a change in the number of objects in a region. The input to this extension are IDT descriptions of temporal intervals \([t_1, t_2]\). For example, we might have one that describes the region-crossing of object OBJ7 through region R31. To find out which other objects are also in this region during this time interval we access the data held in the interval tree. This is done using a function called lookup of type frameno \( \rightarrow \) THB \( \rightarrow \) IDT that returns a list of IDTs describing which objects are in what regions at the given time-cell. We combine the results from lookup to determine the number of objects in R31 during the given interval. This extension works because the IDTs are incrementally extended for each frame update, as long as all such reasoning about the IDTs in the interval tree is done after the database has been updated.
Table 4.3: Closing the door, an example showing events and episodes.

### 3.4 Comparison

In this comparison we first describe a “made-up” close-the-door scenario to illustrate the IDTs that are incrementally created in the interval tree database. After working through this example we briefly compare the approach used here with similar techniques such as the script-based approaches outlined in chapter 2 sections 3.2.1 and 4.3.2.

The close-the-door scenario we are using here is shown in table 4.3. From these values we would like to identify similar events and episode to those described in figure 4.14. The table is a little misleading since it displays all the frames in one go and does not contain any posebox data. Once the intervals are created (with the beginning and ending identified) we can reason with them to describe the activities typing, walk and in-doorway, and the events “typing → walk” and “walk → typing”.

<table>
<thead>
<tr>
<th>time</th>
<th>orientation</th>
<th>heading</th>
<th>speed</th>
<th>occupied region</th>
<th>speed'</th>
<th>orientation'</th>
<th>activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>5</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>5</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>5</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>5</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\( \forall t_1 < t_2 \\
\text{TRUE}(t_1, t_2, \text{TYPING}(x)) \Rightarrow \text{TRUE}(t_1, t_2, \text{SPEED}(x) = 0) \land \text{TRUE}(t_1, t_2, \text{SPEED}'(x) = \text{same}) \land \\
\text{TRUE}(t_1, t_2, \text{ORIENTATION}'(x) = \text{same}) \land \text{TRUE}(t_1, t_2, \text{chair} \in \text{OCCUPIED-REGION}(x)) \)

\( \forall t_1 < t_2 \\
\text{TRUE}(t_1, t_2, \text{WALK}(x)) \Rightarrow \text{TRUE}(t_1, t_2, \text{SPEED}(x) > 0) \land \text{TRUE}(t_1, t_2, \text{walkable} \in \text{OCCUPIED-REGION}(x)) \land \\
\text{TRUE}(t_1, t_2, \text{clear} \in \text{OCCUPIED-REGION}(x)) \)

\( \forall t_1 < t_2 \\
\text{TRUE}(t_1, t_2, \text{IN-DOORWAY}(x)) \Rightarrow \text{TRUE}(t_1, t_2, \text{SPEED}(x) \geq 0) \land (\text{SPEED}(x) \leq \text{moving-slowly}) \land \\
\text{TRUE}(t_1, t_2, \text{doorway} \in \text{OCCUPIED-REGION}(x)) \)

To identify a close-the-door episode as described in figure 4.14 we need some further data concerning the door’s position, and the elevation of the person closing the door (from a ground-plane view it is difficult to describe get-up and sit-down). However, the example illustrates the basic technique. The construction of the episode joining "typing\[\text{walk}\]" and "walk\[\text{typing}\]" is given by:

close-the-door2 = typing \[\text{walk, in-doorway, walk}\] typing

The identification of the pattern described by close-the-door2 provides a conceptual description of the events happening in the "made-up" scenario.

### 3.4.1 Other Computational Approaches

The above example demonstrates the usefulness of the THB approach, however, it is limited in what it can represent. To form more complete conceptual descriptions we require more complex pattern recognition. We reviewed this research area in chapter 2 sections 3.2.1 and 4.3.2. For example, Neumann [173] describes an initial step being the development of an event hierarchy such as that given in figure 4.32 (also see table 2.1 in chapter 2). We can use the events identified in this hierarchy as the basis for a simple context-free language like that outlined in figure 4.33, which makes the temporal ordering explicit like that of the cellwise-time episode representation use in the THB. When applied to application domain input data derived from compact encodings there is likely to be noise contained in the input data, which may cause additional, missing or transformed events. Corral and Hill [46] describe a solution to this problem called "island parsing", which treats recognised episodes as islands and infers from these which of the possible higher-level scripts best fit the identified episodes.

An alternative would be to redescribe the simple context-free language in figure 4.33 as a finite-state-machine, such as that given in figure 4.34. This does not really provide much of an advantage over the THB representation, however, this approach can act as the basis for developing a hidden-Markov model such as that shown in figure 4.35. Hidden-Markov
Chapter 4. Events and behaviour

Figure 4.32: An event hierarchy.

Figure 4.33: The use of partial episodes (PE0, PE1, ...).

Figure 4.34: Event states and connectivity.

Figure 4.35: Two dimensional lattice structure containing all possible state-sequences that can describe the given observation sequence. It shows how recognition using a hidden-Markov model can be considered to be a path searching model.

Models have been successfully applied to speech recognition (Rabiner [192]), a real-world domain that requires the recognition of more complex signals than those used by the THB.

Finally, another approach using Bayesian networks is described by Nicholson and Brady [176, 177] which they call “weights” (see appendix D). The benefit of using a probabilistic approach is that it can express how the identification of a particular episode becomes more certain as more of its events and activities are recognised in sequence.

3.5 Summary

In this section we have discussed the second stage of event-reasoning, describing how events can be composed to form episodes and, that collected together, these episodes provide a simplified form of conceptual description. Although important, the detection of episodes is not as central to this work as the demonstration of how spatial representation and reasoning can be used to describe the activity of the scene objects.

We have addressed the problem of event composition providing a data-driven solution within the passive vision framework described in section 1. Event composition does not involve spatial reasoning, it is only composing previously identified properties, to determine the temporal extent over which they hold. This simple form of temporal representation allows us to identify predefined patterns that describe episodic behaviour. The implementation described here was chosen for its simplicity, although it does make novel use of the
dynamic interval tree for representing the extent of intervals. The computational framework uses a script based approach, similar approaches developed in the VIEWS project [46], and described by Neumann [173].
Figure 4.36: A car entering the roundabout (frame times 450 to 540 in steps of 10).

4 EXAMPLES

In this section we describe two examples which concern entering-the-roundabout where we describe the behaviour of one object and in-the-same-region where we partially describe the behaviour of multiple objects.

4.1 ENTERING THE ROUNDBOUT

As already described, the vehicle path data is a series of ground-plane vehicle outlines from a model matcher. Figure 4.36 shows a sequence of discrete samples from the path of a vehicle as it enters the roundabout which is taken from the "enter scenario" described in appendix A. Here the event history is used to identify region crossing events, and given this functionality, this example is to demonstrate that we can determine what the vehicle is doing. Figure 4.37 shows the conduit of temporally stacked poseboxes, the temporal axis is displayed more clearly in figure 4.38 where we see the 2D+t conduit from the side.

Figure 4.39 shows the results of processing the compact encodings, giving a qualitative description in terms of the labelled front, sides and rear (which are the 1-D cells of the vehicle region). Figure 4.40 contains labelled regions so that the duration of occupancy can be correlated to its location on the ground-plane. In figure 4.39 the regions are in type order, which leads to some repetition, since a leaf region can be part of more than one composite region.

Looking back at figure 4.38, we can see that it gives a visualisation of the region crossing events and, indicates their duration on the vertical time axis forming a 2D+t spatial script. The data contained in this 2D+t representation can be selectively decomposed to obtain a description of how long the vehicle is in each region and a measure of its orientation change. Both of these aspects are shown in figure 4.41 and in the qualitative propositional clauses given below.
Figure 4.41 contains two result windows. The top one describes the region crossing temporal intervals. Here we are forming a region crossing event from the state change of the front edge entering the region and the state change of the rear edge exiting the region (assuming forward motion). As in figure 4.39, partial occupancy is shown by dotted lines (we check to see if the whole vehicle passes through a region, for example, the left-hand side of the vehicle skims the edge of the central-island). This result window tells the story of object13's turning event showing, diagrammatically, how object13 began in the fast-lane of the entry-road, then entered onto the roundabout (by crossing over the cycle-lane), typical of a vehicle turning-right.

The lower window in figure 4.41 shows the quantitative orientation values calculated as differences between each frame. The differences are thresholded, to cut down the noise. The changes in orientation tie up with the spatial region crossings, showing that the maximum orientation change occurs when object13 is crossing from the entry-road onto the roundabout. The figure provides an intermediate presentation of results, which are intended to form the basis for further (more global) reasoning. To facilitate this we use qualitative propositional clauses which have the form true(T P) where T is a time interval and P is a well formed formula.

true( < 450, 490 >, region-crossing(OBJECT13, complete, region(R90), of-types(ENTRY-ROAD FAST-LANE PEDESTRIAN ROAD-SURFACE)) )

true( < 450, 479 >, region-crossing(OBJECT13, partial(LEFT-SIDE), region(R89), of-types(CENTRAL-ISLAND PEDESTRIAN)) )

true( < 450, 459 >, region-crossing(OBJECT13, complete, region(R38), of-types(ENTRY-ROAD FAST-LANE ROAD-SURFACE)) )

true( < 450, 504 >, region-crossing(OBJECT13, partial(LEFT-SIDE), region(R91), of-types(CENTRAL-ISLAND)) )

true( < 450, 510 >, region-crossing(OBJECT13, complete, region(R92), of-types(ENTRY-ROAD TURN-RIGHT FAST-LANE ROAD-SURFACE)) )

true( < 480, 510 >, region-crossing(OBJECT13, partial(LEFT-SIDE), region(R93), of-types(CYCLE)) )

true( < 480, 521 >, region-crossing(OBJECT13, complete, region(R87), of-types(ENTRY-ROAD FAST-LANE TURN-RIGHT CYCLE ROAD-SURFACE)) )

true( < 500, 539 >, region-crossing(OBJECT13, partial(LEFT-SIDE), region(R482), of-types(ROUNDABOUT ROAD-SURFACE)) )

true( < 500, 540 >, region-crossing(OBJECT13, partial, region(R167), of-types(ROUNDABOUT TURN-RIGHT ROAD-SURFACE)) )

true(< 510, 520 >, turn-threshold-exceeded(OBJECT13, orientation-change(RIGHT)) )

5Ideally the orientation information would be provided explicitly by the the intermediate-level visual processing.
Figure 4.37: The conduit from above.

Figure 4.38: The conduit from the side, showing region crossings.

Figure 4.39: The region occupancy of the vehicles front, sides and rear. The transitions where the vehicle only partially enters a region are denoted by dotted lines.
Figure 4.40: The names of the leaf regions (for more details see figure 3.15 in chapter 3).

Figure 4.41: A pictorial description of the predicate data.

The temporal intervals show the length of time that object13 was in a particular leaf region, i.e., the time during which object13 was crossing the region from first entering until finally exiting. In this example, it is the change in orientation (within some predefined allowable range) while the vehicle is continuously in a turning-region that makes this a valid turning event. The above clauses also describe the transition of object13 from entry road to roundabout, all of which takes place on a continuous road surface, allowing us to assert that object13 has legally entered the roundabout. So far we have just provided a simple representation of time. Temporal reasoning, itself, only becomes necessary when we are considering multiple vehicles.
Figure 4.42: The frames. The numbers attached to the poseboxes are the numerical part of the object identifiers supplied by intermediate-level visual processing.

4.2 IN THE SAME REGION

To demonstrate the description of multiple object behaviour we will use the insameregion algorithm given in equation 4.3 from section 2.3. Here the event history of region transitions is used to form region crossing episodes in the THB. The results from these region crossings are used by the insameregion algorithm to form additional intervals that denote when two or more objects are in the same region. As shown in figure 4.42 we will use another frame sequence which is taken from a different section of the enter scenario (see appendix A). Figure 4.43 shows the conduits for all the objects identified in the compact encodings illustrated in figure 4.42. To investigate the behaviour present in this sequence we have isolated the activity of objects 5 and 7 as shown in figure 4.44. Figure 4.45 shows the results from the insameregion algorithm in the form of space time volumes for the durations that objects 5 and 7 are in the same region. These are the wire frame region outlines extended along the temporal axis. The path of the two selected objects begins in regions R482, and travels through R157, and R484 to end in R156. The spatial relationship of these regions is shown in table 4.5. Table 4.4 shows the durations that the pair are together in each of these regions. These results describe in topological terms the durations when the two objects are in the same region. We described in section 2.3 the efficiency of this algorithm and the results do describe that the two objects are on the same road.

The space time volumes are difficult to interpret because they do not reflect the everyday form of this data. The results might be more understandable if a different tesselation of the ground-plane was used, although any static tesselation is going to encounter problems representing the dynamic property of proximity between the scene objects that we are trying to capture here.
Figure 4.43: Four conduits from the overtaking example.

Figure 4.44: Two conduits from the overtaking example.

Figure 4.45: The space time volumes of region occupancy.
Table 4.4: The time intervals during which the objects 1, 5, 7, and 9 are in the same region.

<table>
<thead>
<tr>
<th>interval</th>
<th>region</th>
<th>objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>270 ... 317</td>
<td>R462</td>
<td>5,7</td>
</tr>
<tr>
<td>272 ... 298</td>
<td>R157</td>
<td>1,5</td>
</tr>
<tr>
<td>288 ... 293</td>
<td>R157</td>
<td>1,5,7</td>
</tr>
<tr>
<td>293 ... 305</td>
<td>R157</td>
<td>5,7</td>
</tr>
<tr>
<td>323 ... 360</td>
<td>R484</td>
<td>5,7</td>
</tr>
<tr>
<td>346 ... 360</td>
<td>R156</td>
<td>5,7</td>
</tr>
</tbody>
</table>

Table 4.5: Leaf region names.

4.3 SUMMARY

These implementations have shown that it is possible to make the representation of the vehicle semantics clearer by having a spatial boundary structure that is the same as that used to describe spatio-temporal intervals. This provides a uniform spatio-temporal representation from which spatial and/or temporal information can be extracted as necessary. Using this uniform spatio-temporal representation supports reasoning about multiple vehicles that are in the same region, making explicit how the objects interact (if at all) in both space and time. These examples have demonstrated that SPATIAL-LAYOUT can be used to provide spatial context for identifying valid behaviour.

The main problem with this approach is that it is difficult to extend so that more detailed descriptions of how the various objects are arranged in relation to each other. In these examples, to make the results more understandable we have isolated the compact encodings of the objects that are relevant to the behaviour we are illustrating. Although we have used a simple history mechanism in the form of the THB, it seems likely that regardless of the implementation used we would have a combinatorial explosion if we try to generate all object single, binary and multiple properties. This is typical of the passive, off-line approach to vision in general. A more active, situated approach with selective attention could allow us to perform behavioural evaluation in a timely manner. This will include addressing the decomposition of the behavioural reasoning, coordination of results as well as planning and control issues.
5 REASSESSMENT

In this section we consider the results from HIVIS-MONITOR and measure these against the surveillance problem. The results described in section 4 illustrate the limitations of the first HIVIS-system and our objective here is to identify the important problem areas.

5.1 SCRIPT-BASED APPROACH

As a testbed HIVIS-MONITOR works reasonably well, and it can identify vehicle behaviours such as: turning, giving-way, vehicles-in-the-same-region. We claim that HIVIS-MONITOR demonstrates typical traits of the class of traditional AI approaches we have called "script-based". It is quite likely that better implementations can be developed that fulfill the script-based approach (see, for example, the description of plan recognition in chapter 2 section 4.3.2). However, the exemplar developed here should be enough to judge whether to remain with this approach or to investigate an alternative.

5.1.1 GENERAL FEATURES

In general, all script-based systems will have the following features:

- Maximal detail is derived from the input data. From the stream of compact encodings this approach obtains a description of all objects and all interactions, over the whole scene, for all the episodes it has been designed to detect.
- Representation is extracted first and the results are placed in an evolving database that is used to construct more abstract descriptions using hindsight.
- Single object reasoning is performed with ease using this approach.
- Simple implementation can be achieved using standard AI techniques.

These features are illustrated by the examples in section 4 and cause the following limitations.

5.1.2 LIMITATIONS

HIVIS-MONITOR has the following limitations:

- It is passive in its processing, operating a simple control policy, that is, not affected by changes in the perceived data.
- It is not real-time because the construction of the results database is an off-line process, and does not send feedback to any form of intermediate-level visual processing.
- Unbounded storage is required because any pieces of data contained in the results database might be needed later either to compose some more abstract description or to be accessed by the user to answer a query. Since we do not retract what we have seen or the episodes that we have identified, the database structure is monotonically increasing in size.
• Multiple object reasoning is difficult within the global coordinate system used to express pose positions. A solution to this is needed because as demonstrated in section 2.3, contextual knowledge is not enough to analyse the interactions, although it does provide a context for interpretations.

• The computation performed by HVIS-MONITOR is mainly dependent upon the number of objects in the input data, i.e., it is data-dependent.

• This model is inflexible because it only deals with known episodes. Within the constraints of the predicates provided (language primitives that describe events and activities), new behavioural models can be added. However, defining new predicates may be difficult.

• The addition of new operators increases the number of tests performed on all the objects in the scene. For a single object operator there is a \( O(n) \) increase, for most binary object operators there is a \( O(n^2) \) increase, and for most multiple object operators the increase is polynomial with a maximum of \( O(n^4) \), where \( n \) is the number of objects in the scene.

• The behavioural decomposition does not take into consideration the temporal context in which the events have occurred, which contributes to the process of interpretation. It is possible that the selection of the "correct" episode description is not possible due to only seeing part of an episode.

5.1.3 Discussion

From these features and limitations we can identify the following key problems: computation is performed to obtain results that may never be required; and as the database of results increases in size, the performance of the system will degrade. It might be possible to address these by extending the script-based approach however, we will not take this evolutionary route. Instead we will investigate a more situated approach. This new approach differs greatly from the passive, data-driven script-based approach and requires a complete reformulation of the problem to obtain an active, task-driven situated solution. To begin this reformulation we first consider the use of more local forms of reasoning in terms of the frame-of-reference of the perceived objects, the spatial arrangements of these objects and the use of contextual indexing.

5.2 Frames of Reference

In the spatial representation discussed in the previous chapter, a global extrinsic coordinate system was assumed. By taking a global view we comply with a commonly held Western view of how to represent space in a map-like way. The absolute coordinate system also fits well with the concept of the optic-array (see Gibson [85] and Kosslyn et al. [126] for details), if we can consider placing a grid over the ground-plane to be analogous to the optic-array of the perceiver. An example of this is given in figure 4.46. This representation would allow reasoning to be performed that does not need full object recognition with spatial relationships represented in terms of the optic-array's absolute coordinates (in some respects this is like the video-game world used by Agre and Chapman [3] where
the "winner-takes-all" recognition mechanism (see Chapman [34], Koch and Ullman [122]) allows objects and their positions to be identified by key properties, such as, colour and roundedness).

In contrast to this global viewpoint, when reasoning about the behaviour of each scene object it would be useful if the representation of the properties related to each object could be described in its own relative coordinate system. However this involves recognising each object to the extent that an intrinsic-front can be identified together with its spatial extent. This requirement places the need for a more sophisticated understanding of how the image data present in the optic-array relates to how objects exist in the environment. In the surveillance problem we are given the pose-positions of the scene objects making local reasoning attractive, although its extra cost in terms of the complexity of intermediate-level vision should be noted.

**Definition 4.4** The local-form is representation and reasoning that uses the intrinsic frame-of-reference of a perceived object (exocentric with respect to the observer).

**Definition 4.5** The global-form is representation and reasoning that uses the perceiver's frame-of-reference, which operates over the whole field-of-view (egocentric with respect to the observer).

The global-form is not a public-world since it, like the local-form, only exists to the perceiver. We are not dealing with representing a shared world in terms of each participant.

Some surveillance tasks include both local- and global-form. For example, if we consider a number of objects that are queueing for some reason. As shown in figure 4.46, in the road-traffic domain they might be waiting for a tram to pass. The perceiver of this queue might use local and global reasoning to make different properties of the queue apparent. The local-form refers to each observed object's own deictic frame-of-reference, providing descriptions of the other objects and the local queue properties in these terms. This approach makes it possible to define the queue and queue related activity (join, leave, break-from, jump) task-dependently and robustly. The global-form refers to the queue as a whole, and would be a useful stance from which to count the number of objects in the queue. However, this global viewpoint, as shown by HIVIS-MONITOR, is not so useful for describing the activity of the queue elements.

### 5.3 Spatial Arrangements

The identification of local- and global-forms allows us to reassess the spatial arrangements problem. We need the global coordinate system to provide an invariant metric over the ground-plane, which is used to describe direction of a given straight-line trajectory and from this its velocity (i.e., a directed magnitude). The local-form does not provide any benefits for single-object reasoning, however, once each object's intrinsic front is identified, it provides a natural framework within which to describe the spatial arrangements of objects in the scene. For example, absolute coordinates make it easier to reason about
rectangular objects that are orthogonal to the axes. This can make it appealing to slightly alter the orientation of the objects to make them conform, so that when reasoning about an office space we might limit the possible arrangements of chairs, desks, etc. to those orientations where the faces of the objects are parallel to those of the walls. This use of absolute scene coordinates seems unnatural, although it simplifies the general positional relationships between the various scene objects, it is not suitable for some application domains where the spatial placement of a posebox can contribute to how an object’s behaviour is interpreted. We will investigate how to use the local-form description of an object’s pose to obtain a description of its relation to the other scene objects.

5.4 CONTEXTUAL INDEXING

The region occupancy example given in section 4.2 shows how contextual indexing of the scene objects can be used to identify a form of proximity. This result could then be used to make subsequent computations more relevant and focused. The insamerregion algorithm has used the topological representation present in SPATIAL-LAYOUT to develop space time volumes for an object’s occupancy of a region. Reasoning with this topological representation has reduced the computational complexity because we are dealing with regions rather than the objects themselves.

In addition to using the topological property of the leaf regions, we have shown in section 4.1 how the semantic information present in SPATIAL-LAYOUT can be used to
describe the behaviour of individual objects. Although this data is part of the global-form, by accessing it through the position of the attended object we obtain information that is relevant to the object’s local context.

5.5 Summary

In our reassessment of HIVIS-MONITOR we have identified the problems we are going to address in the remainder of this chapter on events and behaviour. These problems are due to the dynamic nature of the data we are trying to interpret, which has raised issues of uncertainty, immediacy, continual change and situatedness. HIVIS-MONITOR has shown how important they are, even for a simple testbed that is demonstrating the computational feasibility of SPATIAL-LAYOUT. Table 4.6 lists the differences between the two HIVIS-based systems we develop in this dissertation.

The suitability of each HIVIS-system is described in Table 4.6, providing an indication of the extent of the reformulation to the surveillance problem. HIVIS-MONITOR would be useful for off-line query of behaviour, whereas in HIVIS-WATCHER, by asking the question first, we remove the importance of the results database because we are no longer providing a query-based system. This removes the need to remember everything and solves the problem of the monotonically increasing database. The development of a more situated approach in HIVIS-WATCHER is part of the adoption of a more local viewpoint that uses a deictic representation of space and time that draws on the work we described in chapter 2 sections 3.2.4 and 4.2 where we identified reflexive accountability and context as being important features.

It is not yet clear whether using the HIVIS-MONITOR approach of processing all scene objects is necessary. This is an issue we will consider again and resolve in section 8 when we conclude this chapter. To reduce the computation of unnecessary results we will explore the use of tests, such as contextual indexing described in section 5.4, to make computations more relevant and focussed.
 CHAPTER 4. EVENTS AND BEHAVIOUR

<table>
<thead>
<tr>
<th>entity</th>
<th>long form</th>
<th>abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>aspect</td>
<td>the-reference-object-I-have-selected</td>
<td>the-refobj</td>
</tr>
<tr>
<td></td>
<td>the-reference-object-I-have-selected-is-moving</td>
<td>the-refobj-is-moving</td>
</tr>
<tr>
<td>entity</td>
<td>the-secondary-object-to-refobj</td>
<td>the-other</td>
</tr>
<tr>
<td>aspect</td>
<td>the-secondary-object-to-refobj-is-moving</td>
<td>the-other-is-moving</td>
</tr>
</tbody>
</table>

Table 4.7: Simplifications using the-refobj and the-other.

6 ENTITIES AND ASPECTS

We will use the terms “entities” and “aspects” to describe the deictic representation used here. Both of these terms are taken from the work of Agre and Chapman [2, 3, 34].

**Definition 4.6** An entity is something that is in a particular relationship to the agent.

**Definition 4.7** An aspect describes a property of an entity in terms of the agent’s purpose.

Aspects provide information about the activities of an object, not its events. For example, the-cup-I-am-drinking-from is the name of an entity and the-cup-I-am-drinking-from-is-hot is the name name of an aspect of it. Each time we obtain an aspect (say by picking up the-cup-I-am-drinking-from) its value may be different. We still have the problem of identifying events from the temporal sequence of aspect values (such as the event when the-cup-I-am-drinking-from-is-hot becomes false) but this is made more complex because the entity is not always the same (for example, during a typical day, on different occasions, I drink from a number of different cups). However there is some local temporal continuity.

This framework works well when we describe things in relationship to ourselves, and in the video game context where we are personified by the character/token that we control. In the surveillance problem we do not have this direct contact with the observed objects, yet we still need to employ our knowledge of how we would typically act in that situation to enable us to partially understand what is happening. To begin this process we apply some form of the aspects we would employ in the same situation to describe the properties of the object we are attending. For example, when observing a person who gets up to close the door, we use knowledge of how we have closed doors ourselves to understand the perceived activity and the reasons for it. Although experience of having performed an observed act is not essential, we need to make analogies to previous experience, with greater similarity likely to increase understanding, if only because it brings with it knowledge of what to look for.

In section 1.2 we introduced the common primitives, which provide aspect like properties about single objects, here we want to apply the local-form to obtain aspects about the other objects in relation and relative to an attended object. We need to extend the deictic approach so that we can refer to scene objects, which are entities to the official-observer, and to the other scene objects that have a relationship to our primary reference object.
We will call this reference object *the-refobj* which is short for *the-reference-object-I-have-selected* where I refers to the official-observer. We also use the abbreviation *the-other* as described in Table 4.7 to refer to these secondary objects of interest.

The problem with this extension is that if we want to obtain all binary relationships between each pair of objects that *the-refobj* and *the-other* can point to in the scene, then we have an $O(n^2)$ number of tests to perform for each particular relationship.

In the first part of this section we describe some deictic relationships we will use in chapter 5. After discussing the usefulness of deictic representation for describing binary spatial arrangements we then discuss how the problem of obtaining all object relationships can be reduced. We finish this section with a description of how entities and aspects can be used in a peripheral system (which is the input part of Fodor’s input/central split described in Chapter 2 section 4.3.5 on modularity).

### 6.1 Deictic Primitives

Here we return to the problem given in Figure 4.20 of how to describe spatial arrangements between two objects. To investigate this problem we will use a set of qualitative orientations called numθ consisting of eight 45° rotation steps as shown in Figure 4.47. There are a number of possible relationships for each pair of orientations illustrated by the grid cells shown in Figure 4.48 which depicts the Cartesian product of the set of numθ for *the-refobj* and the set of numθ for *the-other*. Each relationship pair only defines the orientation of each object in a global coordinate system, without describing their positional correspondence with regard to each other. This numθ grid provides a framework in which to consider the spatial arrangements problem although it does not solve it. The global reference frame of Figure 4.49 appears promising, it illustrates how eight options can be used to resolve
the positional relationship between the-refobj and the-other. In figure 4.49 the-refobj is represented by the grey circle in the centre of the figure and the-other assumes one of the places labeled \( \{a, b, c, \ldots, h \} \). When we illustrate one instance of the set of global positions in figure 4.50 where the-refobj has orientation 2 and the-other has orientation 3 we show the problems with the global approach. There is redundancy present in the qualitatively exhaustive description of binary spatial arrangements and we have not provided a useful mapping to terms that describe spatial arrangements like behind, left, right and infront we identified in chapter 2 section 3.2.3. We address this problem next under the heading “positional” and then consider other binary object relationships.

6.1.1 Positional

In HIVIS-WATCHER we have implemented the “basic order” frame-of-reference introduced in chapter 2 section 3.2.3, although we tend to ignore the left/right distinction by using “beside”. The positional locations formed by the field values of an object are used to cut space into qualitative regions. For example, in figure 4.51(a) we have a reference object \( K \). In parts (b) and (c) we describe the position of some object \( A \) that is to the right of \( K \) by presenting two different interpretations with the shaded area in each part of the figure showing where \( A \) can validly be: in (b) if it is within a 90° angle and in (c) the right hand half plane. When we apply these models to our poseboxes, we introduce the pose-position of the reference object, but continue to assume an idealised situation where the other objects are points.

Figure 4.53(a) shows the initial intuitive model, the edges of the object are combined to form not only the region of space that encloses the object, but also eight external spaces labelled f,b,l,r,fl,fr,bl,br. The four hyperplanes, individually highlighted in figure 4.52, are oriented manifolds, having a negative and positive side, in the figure we have shaded in each of the positive half planes to illustrate the area for front, left, right and back. Details of how this can be implemented are given in appendix C section 3.1. The four hyperplanes are used to label the position of any point in terms of this qualitative coordinate system.

The half-space model described in figure 4.52 and figure 4.53(a) produces some unexpected results, which were due to overestimating the range. Figure 4.53(b) resolves this problem by adapting figure 4.51(b). Figure 4.53(b) shows the reformulation of the half-space model, with the “f” area reflecting the visual-field infront of the agent. In figure 4.53(b) location f is like figure 4.51(b), and locations fl, f, fr together are like figure 4.51(c).

This qualitative coordinate system is used to obtain values for the aspects the-other-is-infront, the-other-is-behind, the-other-is-on-the-left, the-other-is-on-the-right, and the-other-is-beside. These aspects are illustrated in figure 4.54. In figure 4.55 we have three objects described by their poseboxes and half-space models. The objects are labelled a, b and c. In figure 4.56 we consider object b, which has the aspects the-other-is-behind-and-left and the-other-is-infront where each aspect refers to a different object. In addition to this ambiguity, the figure also demonstrates that there is not a clear inverse relationship
Figure 4.51: Location outside the reference object K. A is to the right of K. From [98, page 181].

Figure 4.52: The four half spaces.

**Figure 4.53: The half space model.**

<table>
<thead>
<tr>
<th>POSITIONAL ASPECTS</th>
<th>ILLUSTRATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-other-is-behind</td>
<td></td>
</tr>
<tr>
<td>the-other-is-on-the-left</td>
<td></td>
</tr>
<tr>
<td>the-other-is-beside</td>
<td></td>
</tr>
<tr>
<td>the-other-is-on-the-right</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.54: The positional aspects of the-thereobj.

Figure 4.55: The half spaces of three cars.

Figure 4.56: Pick a reference object.
Figure 4.57: Heading difference between the-refobj and the-other.

<table>
<thead>
<tr>
<th>refobj</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Heading Difference</th>
<th>Same Heading</th>
<th>At Right Angles</th>
<th>Opposite Heading</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Theta$</td>
<td>$0 \leq \Theta &lt; 1$</td>
<td>$1 &lt; \Theta &lt; 3$</td>
<td>$3 \leq \Theta \leq 4$</td>
</tr>
</tbody>
</table>

Figure 4.58: The same heading. (a) is parallel ($\Theta = 0$), and (b) is diverging ($0 \leq \Theta \leq 1$).

<table>
<thead>
<tr>
<th>$\Theta$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4.59: Heading difference values for the operator $\Theta$.

between two objects, for example, (X infront Y) is not necessarily the same as (Y behind X).

### 6.1.2 Heading Difference

In addition to "position" the deictic primitives are used to determine relative values for the heading of the other objects in relation to the-refobj. For example see figure 4.58.

Figure 4.50 illustrates how heading and orientation are independent of relative position, this is reflected in the terms we use to describe the difference in heading between our two indexed objects. The aspect the-other-has-same-heading is true when the-other has a similar heading to the-refobj. The aspect the-other-is-at-right-angles when the-other's heading is ±90° from that of the-refobj. The aspect the-other-has-opposite-heading is true when the-other's heading is ±180° from that of the-refobj. Figure 4.59 provides values for the num$\Theta$ Cartesian product that are used to describe the three heading aspects. The arguments to $\Theta$ are ($\Theta$) :: num$\Theta$ → num$\Theta$ → num and we can define the function ($\Theta$) as being ($\Theta$) $a \ b = \text{abs}(a - b)$. The results from ($\Theta$) for positions of the-other with respect to the-refobj are given at the bottom of figure 4.57 describing how the range of ($\Theta$) can be interpreted as three qualitative values.

This is used to describe when the-other-is-parallel, the-other-is-at-right-angles, or the-other-is-head-on. The approach uses the heading property (from table 4.2) of the two objects, and the orientation property if the heading is not available (e.g., because one or both the objects is stationary or has just entered the scene). Heading is preferred since it is using the direction of motion and can take account of unusual situations such as reversing or walking backwards.
6.1.3 Speed difference

To determine the relative motion of the-other object in relation to the-refobj we use the values for single object speed described in table 4.2. This is basically the subtraction of the the-other’s speed from that held by the-refobj. The results provide values for the aspects the-other-has-same-speed, the-other-is-faster, and the-other-is-slower.

6.1.4 Discussion

We have described how, for poseboxes in a 2D ground-plane, the classes of useful deictic primitives includes position, heading and speed. The problem with these deictic primitives is the number of interrelationships that need to be resolved is polynomial in time complexity.

The benefit of the local-form is that it has provided a qualitative solution to part of the spatial arrangements problem. The local field values of an object are not used to determine heading and speed differences, so these properties are not as dependent on the local-form as the positional model. The positional model is central to the deictic approach developed here, and this deictic framework has made clearer the mechanism of describing the world (e.g., the-other objects) in terms of a selected object (e.g., the-refobj).

Extending the spatial representation to use these deictic relationships raises the problem of how to store and reason about all the interrelationships between the various scene objects. A large and complex pattern matching database might be required to manage this problem. However, for most surveillance tasks attending all objects is unnecessary. To provide this attentional approach we introduce gestalt properties that can be used to guide the selection of which objects to attend. We introduced this idea in section 5.4 in our discussion on the use of contextual indexing.

6.2 Gestalt primitives

Some of the interrelationships that we wish to describe are similar to the early discoveries made in Gestalt psychology. Gordon [89, pages 46–75] provides an introduction to the work of Wertheimer [250] and others (also see Beardslee and Wertheimer [16]), describing how the gestalt grouping properties follow the basic theme of the whole being different from the sum of its parts. There are a number of grouping properties, such as: proximity, similarity, good configuration, common fate (spatial and/or temporal continuity), closure, and symmetry. Some of these are illustrated in figure 4.60.

The gestalt primitives are used to restrict the use of the deictic primitives above to one pairing per reference object, which greatly reduces the computation performed, we require a test to make sure that the selected pairing is the best. We can also use the result of the test to select which objects to attend, because even the computation of the one binary relationship may be unnecessary for some the-refobj’s. In this respect the gestalt primitives can act as primitive cues in an attentional mechanism with the test applied to all objects in a global way over the whole scene.
Figure 4.60: An illustration of Wertheimer’s Laws of Grouping: (a) Proximity induces grouping by rows, (b) proximity is equal and there is no dominant direction of grouping, (c) grouping by continuation, and (d) grouping by similarity. From Gordon [89, page 54].

Rock [202] and Treisman [234] both describe the proposal of attributing visual processing to two stages called “preattentive” and “attentive”. At the first stage simple features are preattentively registered. At this stage global features are described but fine detail is not. Treisman provides examples of texture segregation (a prerequisite for figure ground separation). At the second stage objects are identified using the candidates set up by the preattentive stage. (We described other properties of attention chapter 2 section 4.2.3.) Mahoney and Ullman [149] describe how the results from demonstrations of the preattentive stage support their use for directly indexing local features as long as the figure of interest is distinguished from irrelevant figures by a single one of these features. In this way preattentive processing can propose figures for use by attentive processing. Murray et al. [170] discuss other cue like processes such as one to detect “looming motion”.

The computation of the gestalt primitives takes place in the global-form, does not require detailed information about the scene objects and should not be complex. We will describe primitives for proximity, discontinuity and clustering.

### 6.2.1 Proximity

One of the most elementary spatial primitives in spatial reasoning concerns the relative nearness of one object to another. We use a proximity measure as a simple operator which describes how objects generally interact with those that are nearby. The approach used to select the nearest other object is based upon an approach used in robot motion planning (see Latombe [135]). Figures 4.61 and 4.62 show the repulsive field that conceptually surrounds each object. The height of the field reflects the nearness of the proximity relationship. The higher the value the nearer the other object. The nearness measure consists of five qualitative values: not-near, nearby, close, very-close, touching. These qualitative values are based on a distance metric scaled to give an exponentially higher value to nearer objects and discretised to the integer range [0, ..., 4]. Details of how this is implemented are given in appendix C section 3.2. The relationship to Gestalt concepts is the increased interest in scene elements that form groups.
6.2.2 Discontinuity

In contrast to the gestalt primitive "common fate" the preattentive cue of discontinuity identifies the distinguishing property of a change in spatio-temporal continuity. This primitive is not part of Gestalt psychology because instead of identifying a group we are identifying when a figure changes between groups. In the scene, an object may become distinguishable for a short time when it changes one of its properties and so changing membership from one group to another (for example, from "figure" to "ground"). In the scene under surveillance the moving objects represent a group and when one object stops moving to become part of the static background its spatio-temporal discontinuity distinguishes it from the other moving objects for a short time. This change in expected behaviour also makes objects that begin moving distinguishable. Spatio-temporal discontinuity such as the gross change in motion between moving and stationary and visa versa. This can be implemented by using the single object aspects that describe the speed of each object, and the $c^2$ changes for stop and start.

6.2.3 Clustering

This primitive makes use of the grouping properties described above. Work by Watt [246] (also see Humphreys and Bruce [109, pages 167-169]) describes how early visual processes may package together neighbouring items into groups. This allows us to attend to one object, where the object is the group, providing a coding of objects that are contained in the group at different spatial scales. Selecting targets from a group may be time consuming and liable to interference from distractors. There is related work by Pylyshyn [190] who describes how his FINST approach could use hierarchical grouping. Figure 4.63 illustrates three metrics that we could use to identify groups of objects. In addition to providing a measure of proximity the insameregiion algorithm also gives an indication of possible groups that are travelling in the same direction. We do not fully develop a grouping algorithm/model here, although we do illustrate how such functionality could be applied in chapter 5.
6.2.4 Discussion

These gestalt primitives provide the preattentive cues that are used in chapter 5. Use of gestalt primitives and deictic primitives together requires a more complex control mechanism than that described for HIVIS-MONITOR. In chapter 5 we extend the task-and strategic-layer of the surveillance problem by adding attentional control to HIVIS-WATCHER. This control mechanism is needed to reduce the complexity due to the addition of a number of new operators. For most surveillance tasks we only require the use of a subset of the available operators. The reduction in complexity is not present in the operator (i.e., the deictic primitives), it is present in how the operators is used. For example, we can solve the problem of ambiguous aspects by dealing with the nearest object and only considering other objects in special cases. In HIVIS-WATCHER entities and aspects replace the event-detection stage of HIVIS-MONITOR (see figure 4.3) with a peripheral system which consists of operators that can be selected to obtain the aspect values of indexed entities.

6.3 Peripheral System

Here we return to the event detection stage described in section 2 and develop a different approach that adapts Ulman's [243] visual routine processor to provide part of an event detection stage. The description of the visual routine processor follows that used by Agre [2] and Chapman [34]. Both HIVIS event detection approaches employ a set of operators, however, the key difference is that in the HIVIS-WATCHER peripheral-system operators are not run all the time, they are only run when selected by the task-layer control system. The discussion here is separated in two with the first part describing our version of the visual routine processor, and the second part outlining a collection of example operators used by the peripheral-system in HIVIS-WATCHER.

6.3.1 Operator Processor

The operator processor (OP) is a collection of operators (see definition 4.3) that are all given arguments and executed in parallel (i.e., there is no dependence on sequential execution). We will call this "OP-execution". Each operator performs a simple operation that in itself does not do much, so that all the operators in OP should complete quite quickly. The usefulness of the OP becomes apparent when it is coupled to another system that
repeatedly and sequentially performs OP-executions, with the operators in OP given new arguments for each OP-execution.

As described in definition 4.3 the general form of each operator in OP is \((\text{op}_i \ \text{arg}_1 \ldots \text{arg}_n)\), where \(\text{op}_i\) and \(\text{arg}_j\) denote domain relevant symbol names. The last argument position can be given special meaning if it has the name doit? denoting a boolean flag. When present, it states that \(\text{op}_i\) is selectable and is only to be executed when doit? is set to true. When the doit? flag is not present as the last argument then \(\text{op}_i\) is always run on each OP-execution.

The operator \(\text{op}_i\) is defined by self contained set of instructions and does not return a result. Instead \(\text{op}_i\) side-effects the execution environment by setting specially allocated memory addresses as well as common areas of memory. Figure 4.64 shows such an operator, how it gets its arguments from the world representation and CENTRAL-SYSTEM, and how during its execution it can side-effect state, and how the operator's results are sent as output to the CENTRAL-SYSTEM. In the specially allocated memory, called “aspects”, a selectable operator typically has two addresses called “*\text{op}_i*” and “*\text{register-\text{op}_i}*”. The address called “*\text{op}_i*” is given the return value from operator \(\text{op}_i\)'s execution, usually an integer or a boolean, and “*\text{register-\text{op}_i}*” is set to say that “*\text{op}_i*” has been given a value. In a truly parallel implementation a “*\text{op}_i*-ready*” would also be needed to say when the operator has completed. The set of aspects from the PERIPHERAL-SYSTEM have an injective (one-to-one) mapping to the set of wires registered by the CENTRAL-SYSTEM (this can be described by the function \(B : \text{aspects} \rightarrow \text{wires}\)). There is also a corresponding a mapping from CENTRAL-SYSTEM effector-commands to PERIPHERAL-SYSTEM operators which we describe in more detail in chapter 5.

These operators do not form a nice functional language because of their use of side-effects, which can also make them difficult to define and debug. The OP-execution environment is in effect shared by all the operators with this environment passed onto future executions and leaving a thread of environments as the OP-execution history. It needs to be like this to allow situation driven processing. In chapter 5 we describe a CENTRAL-SYSTEM to which OP can be coupled to provide a closed-loop control.
6.3.2 Design decisions

Here we discuss two constraints upon the description of operators in HIVIS-WATCHER. The first constraint concerns the amount of computation that needs to be performed by each OP-execution, which is dependent upon the frame rate of the stream of compact encodings. If we hold the total amount of computation performed by HIVIS-WATCHER constant, then the amount of computation that needs to be performed for each OP-execution is proportional to the time between frame updates. Large temporal intervals between frame updates mean that more computation needs to be performed by the operators and more assumptions and interpolations made about the data. The second constraint is the parsimony of having a small set of operators that can act upon the available data. Chapman [34] provides some examples of primitive elements like lines, rays and activation-planes, and has operators for things like: follow a line; shade a region; pick out the red bit and put a marker on it; and tell whether the red bit is moving. As described in section 1.2, we are not using these low-level visual primitives, but instead use a set of common primitives.

The frame-rate of the data supplied by intermediate-level vision is not sufficiently fast that we can define very simple operators. Some operators need to do a number of "steps" so that the system can generate timely results. These operators are more like micro-routines which detract from the ideal expressed by the second constraint although we can build our micro-routines from less complex primitives.

To retain consistency with HIVIS-MONITOR we did not change the frame rate. The interframe spacing of between 400–500ms provides time for computation to be performed between updates. During development there was the option of allowing the system to perform more than one clock cycle per frame-update, but this raises two problems, the first of how many extra cycles to allow, and the second of how to adapt the operators to cope with unchanging input data. In HIVIS-WATCHER we perform one OP-execution for each frame update. Although not used in the experiments described in this dissertation, a better solution would be to "in between" the posebox data, and so provide the compact encoding data for additional time cells. This solution does not affect the structure of HIVIS-WATCHER, however, to make use of the increase in density of frame updates would require making each operator invocation perform less, the addition of more operators to describe the decomposed functionality, and the addition of more rules to describe the more detailed routines that would be needed. The identification of the level of decomposition that provides an equivalence between OP-execution and "steps" in the definition of routines is left as future work.

6.4 Example operators

The collection of operators that we describe here access the static and dynamic data described in section 1.2 with the objective of providing more useful abstractions of the available data. The operators themselves are not able to store information, it has to be held in external memory where it can be managed as a limited resource. One example
of this is the path information held about each object marker. Some spatio-temporal reasoning operations are continuous, such as tracking, while others, like prediction, only need to be performed when the situation demands it.

There is a correspondence between the use of markers and operators. We introduced markers in chapter 2 section 3.4 when we described tracking, as a mechanism for indexing scene objects. Associated with each marker is a set of rules (which we describe further in chapter 5) that select which operators to run on the object indexed by the marker, with the marker’s “name” passed as an argument to the operator. In some of the examples used here we denote the supplied parameter of such a marker by the argument self. Markers provide a mechanism for reducing the number of objects in the scene for which properties are obtained. Different markers may use different operators, which provides an example of how expectation can affect interpretation.

In chapter 5 we develop the central-system in two stages. The first stage, called “agency”, concentrates on the use of one form of marker called “agent”, so that all agent markers have the same set of rules. The second stage, called “kernel”, uses additional forms of marker, each having a different set of rules.

6.4.1 Agency operators

We separate the primitives into single, and binary object arities, where the binary arity refers to operators that use local deictic references.

Single object primitives

(find-velocity? self doit?) returns the speed, \( s \in \text{num}\beta \), of self.

(find-orientation? self doit?) returns the orientation, \( \theta \in \text{num}\theta \), of self.

(find-occupiedregions? self doit?) accesses \text{spatial-layout} to determine which regions the object is occupying. The results are used to set wires like *turning*, *giveaway* and *roundabout* that report region type information.

(find-nearest? self doit?) sets the wire *the-other* to the name of the entity that is the nearest object.

(find-orientation-change? self doit?) returns the difference in orientation between the current and previous value.

(publicize-stationary-object self doit?) sets the wire *registered-stationary-object* to *t* and the *stationary-object* to the agent’s name.

(change-motion-prior! self doit?) changes the motion-prior of the object’s trackingState.

Binary primitives

(find-speed-difference? self the-other doit?) returns the speed difference between self and the-other.
Table 4.8: Some of AGENCY's wires for which the AGENCY operators obtain aspects and the sections they are defined in.

(find-heading-difference? self the-other doit?) returns the heading difference between self and the-other.

In the current implementation we drop the self argument as effector commands are indexed by agents making the passing of the self parameter unnecessary. For reference, table 4.8 presents some of the wires set to aspect values together with the sections where we describe the operators that obtain the aspects for the selected entity or entities.

6.4.2 Kernel Operators

These operators are similar to those described by Agre and Chapman, they are slightly more low-level than the operators used by the AGENCY markers.

Marker Based Operators These allow the official-observer to place markers in the global frame and perform tests on marker properties.

(warp-marker! m n) move marker m to location of marker n.
(track! m n doit?) move marker m to location of marker n and track the object at this location.
(marker-m-assigned?) is true when m is unassigned. There is an operator for each marker that inspects the data “under” the marker to see if it is “on” an object.

(markers-coincident? m n doit?) tells when the distance between two markers is zero.

Activated regions Activated-regions are used to represent conflict-areas, region-based-predictions, etc..

(marker-in-activated-region? m r doit?) a predicate on a marker m and an activated region r that is true when m is in r.

(reset-activated-plane! r doit?) resets r so that it can be used again.

Figure 4.64 illustrates the globally accessible activation-region store. All references to the store are performed via descriptors so that the control-system only has to pass the integer descriptors as values on wires.

6.5 Summary

In this section we have introduced deictic and gestalt primitives, and described how they can act as attentive and preattentive processes with the preattentive stage being used to cue the use of the attentive stage. The definition of these two stages requires a mechanism that can support selective application of operators. We have described how the peripheral-system can provide selective functionality, and we have provided some example operators that will be used in the description of attentional control in chapter 5. The results from the aspects returned by the application of the various operators describe activities, and via the rules in the central-system can be used to identify the restricted class of events the consist of differences between two successive states.
Chapter 4. Events and behaviour

7 Situations and routines

The difference between the identification of episodes in this section and the description given in section 3 is that here we are combining events that are identified as being relevant to the official-observer, whereas in section 3 the individual events were relevant to the individual, or various groups of scene objects. This reduces the amount of computation necessary to form conceptual descriptions. However, once an event or episode has been identified we still have the problem of how to build more abstract descriptions from these primitives. In section 6.1.1 we derived the deictic relationships from one object's frame-of-reference, combining the information is more difficult. We need to integrate all reasoning from the point-of-view of the official-observer (data association) and not from each scene object's point-of-view. In this section we describe how local positional descriptions are combined to provide a solution to the spatial arrangements problem for two objects. We do this by combining the qualitative descriptions of relationships as seen by each member of the binary relationship.

7.1 Routines

Routines in HIVIS-WATCHER are not just about the scene objects, they also concern the behaviour of the perceiver. We discuss this subject more fully in chapter 5 when we consider control issues. Here we are extending the work on event-reasoning described in sections 2 and 3, which were about the identification of events and behaviour of objects in the field-of-view. The cellwise-time descriptions of episodes used in section 3 do not describe the operator sequence that comprise a routine, instead they model the sequence of event or activity elements that a routine would endeavour to identify. The difference is that a routine contains additional control information that describes how actions are performed, and when they are relevant. Here we will ignore these details concerning control in order to maintain a continuity of representation with HIVIS-MONITOR. Also, once identified the event and activity elements are not used by routines, because they are really side-effects, generated as part of routine performance (possibly by additional operators that are just present to identify the presence of a particular combination of aspect and interval values). This separation corresponds to Newtson's [175]) experimental results described in chapter 2 section 3.2.2 that people normally segment behaviour into actions and are remarkably unaware of their segmentations.

Routines do not have an explicit notion of time that is suitable for describing some of the temporal orderings given here such as the sequential temporal orderings (SEQ) although it is able to reason about simultaneous properties (SIM). We use this in the solution of the spatial arrangements problem to integrate the deictic viewpoints of selected objects. To perform this temporal and deictic integration a separate control component is used to collect the results together, which is described in chapter 5.
7.2 Examples

The examples discussed here illustrate how the deictic primitives introduced in section 6 can be used to define other-overtakes, other-following, refobj-in-queue, refobj-giving-way, other-recedes and other-approaches. Notice that the use of deictic references does not use explicit naming of objects.

7.2.1 Following and Overtaking

Following and overtaking are included in the class of episodes that involve two objects which are proximate to each other and are both travelling in the same direction. This class of episode is used to demonstrate how HIVIS-WATCHER solves the spatial arrangements problem. In section 6 we described how the deictic positional events for other-is-behind, other-is-infront, etc., are obtained. Here we describe how these can be combined, by using the interpretations \textit{SEQ} and \textit{SIM} introduced in section 3. In table 4.9 we take advantage of the symmetry present under \textit{SIM} interpretation and only need to specify ten options instead of all sixteen. In contrast \textit{SEQ} interpretation shown in table 4.10 is not symmetric. This table simplifies the problem of composition by not taking account of partially observed events. We do not use the \textit{SEQ} interpretation in HIVIS-WATCHER because of the limited temporal reasoning present in routines. Extending routines to include \textit{SEQ} interpretation is not easy as there is the likelihood of reintroducing the limitations we identified in the script-based approaches and which this HIVIS approach is trying to solve.

We can describe overtaking and following as:

\[
\forall t_1 < t_6 \\
\text{TRUE}(t_1, t_6, \text{OTHER-OVERTAKES}) \Rightarrow \\
\exists t_1 < t_2 < t_3 < t_4 < t_5 < t_6 \\
\text{TRUE}(t_1, t_2, \text{OTHER-BEHIND}) \land \text{TRUE}(t_2, t_3, \text{OTHER-BEHIND}) \land \\
\text{TRUE}(t_3, t_4, \text{OTHER-BESIDE}) \land \text{TRUE}(t_4, t_5, \text{OTHER-BESIDE}) \land \\
\text{TRUE}(t_5, t_6, \text{OTHER-INFRONT}) \land \\
\text{TRUE}(t_6, t_6, \text{OTHER-INFRONT})
\]

\[
\forall t_1 < t_6 \\
\text{TRUE}(t_1, t_3, \text{OTHER-FOLLOWING}) \Rightarrow \\
\exists t_1 < t_2 < t_3 \\
\text{TRUE}(t_1, t_2, \text{OTHER-BEHIND}) \land \text{TRUE}(t_2, t_3, \text{OTHER-BEHIND})
\]

These descriptions of overtaking and following are illustrated in figure 4.65. We can model a other-overtakes episode from five temporally ordered deictic event primitives [other-is-behind] [other-is-behind-and-beside] [other-is-beside] [other-is-beside-and-infront] [other-is-infront]. The \textit{SIM} conjunction of deictic references that point to the same thing is needed because we are not dealing with a subject that is a spatial point, we are using a posebox that has spatial extent, a front, sides, and
Table 4.9: The table above shows the composition rules that use SIM. The figure below illustrates the arrangements of the objects with the numbers corresponding to the numbered rule definitions. The poseboxes with arrows are moving/heading the blocked in grey ones are blocking-vehicles.

<table>
<thead>
<tr>
<th>VIEWPOINTS</th>
<th>COMPOSITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 other-is-behind</td>
<td>other-is-behind</td>
</tr>
<tr>
<td>2 other-is-behind</td>
<td>other-is-beside</td>
</tr>
<tr>
<td>3 other-is-behind</td>
<td>other-is-infront</td>
</tr>
<tr>
<td>4 other-is-behind</td>
<td>other-blocks-path</td>
</tr>
<tr>
<td>5 other-is-beside</td>
<td>other-is-infront</td>
</tr>
<tr>
<td>6 other-is-beside</td>
<td>other-blocks-path</td>
</tr>
<tr>
<td>7 other-is-beside</td>
<td>other-is-beside</td>
</tr>
<tr>
<td>8 other-is-infront</td>
<td>other-blocks-path</td>
</tr>
<tr>
<td>9 other-is-infront</td>
<td>other-is-infront</td>
</tr>
<tr>
<td>10 other-blocks-path</td>
<td>other-blocks-path</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t_n$</th>
<th>$SEQ$</th>
<th>o-is-behind</th>
<th>o-is-beside</th>
<th>o-is-infront</th>
<th>o-blocks-path</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{n-1}$</td>
<td>o-is-behind</td>
<td>o-follows</td>
<td>o-overtakes</td>
<td>error</td>
<td>error</td>
</tr>
<tr>
<td>o-is-beside</td>
<td>o-overtaken</td>
<td>o-still-beside</td>
<td>o-overtakes</td>
<td>o-swerves</td>
<td></td>
</tr>
<tr>
<td>o-is-infront</td>
<td>error</td>
<td>o-overtaken</td>
<td>o-followed</td>
<td>o-infront-stops</td>
<td></td>
</tr>
<tr>
<td>o-blocks-path</td>
<td>error</td>
<td>error</td>
<td>o-moves</td>
<td>o-still-blocks</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.10: Composition rules that uses $SEQ$. Note that “other” is abbreviated to “o”.
rear. In overtaking, the use of other-is-beside is too weak. A slightly stronger other-is-beside-and-has-same-heading operator would be better making the requirement of parallel-motion explicit.

7.2.2 Queue

In this description of a queue we are taking refobj's viewpoint. From this perspective a queue can be described in terms of being blocked by a stationary object for more than two time-cells.

\[
\forall t_1 < t_3 \\
\text{TRUE}(t_1, t_3, \text{REFOBJ-IN-QUEUE}) \Rightarrow \\
\exists t_1 < t_2 < t_3 \\
\text{TRUE}(t_1, t_2, \text{OTHER-BEHIND}) \land \text{TRUE}(t_1, t_2, \text{REFOBJ-PATH-BLOCKED}) \land \\
\text{TRUE}(t_2, t_3, \text{OTHER-BEHIND}) \land \text{TRUE}(t_2, t_3, \text{REFOBJ-PATH-BLOCKED})
\]

7.2.3 Giveaway

A giveaway episode can be described in terms of refobj being stationary in a giveaway region such that its path is blocked by the object it is giving way too.
\[ \forall t_1 < t_3 \\
\text{TRUE}(t_1, t_3, \text{REFOBJ-GIVING-WAY}) \Rightarrow \\
\exists t_1 < t_2 < t_3 \\
\text{TRUE}(t_1, t_2, \text{REFOBJ-IN-GIVEWAY-ZONE}) \land \text{TRUE}(t_1, t_2, \text{REFOBJ-IS-STATIONARY}) \land \\
\text{TRUE}(t_2, t_3, \text{REFOBJ-IN-GIVEWAY-ZONE}) \land \text{TRUE}(t_2, t_3, \text{REFOBJ-IS-STATIONARY}) \land \\
\text{TRUE}(t_2, t_3, \text{REFOBJ-PATH-BLOCKED}) \land \text{TRUE}(t_3, t_4, \text{REFOBJ-PATH-BLOCKED}) \]

7.2.4 Approach and Recede

We can describe other abstract terms such as recede-from-me and approach-me using lower-level aspects such as faster-than-me, infront-of-me and behind-me.

\[ \forall t_1 < t_2 \\
\text{TRUE}(t_1, t_2, \text{OTHER-RECEDES}) \Rightarrow \\
\exists t_1 < t_2 < t_3 \\
\text{TRUE}(t_1, t_2, \text{OTHER-IS-FASTER}) \land \text{TRUE}(t_1, t_2, \text{OTHER-IS-INFRONT}) \]

\[ \forall t_1 < t_2 \\
\text{TRUE}(t_1, t_2, \text{OTHER-APPROACHES}) \Rightarrow \\
\exists t_1 < t_2 < t_3 \\
\text{TRUE}(t_1, t_2, \text{OTHER-IS-FASTER}) \land \text{TRUE}(t_1, t_2, \text{OTHER-IS-BEHIND}) \]

7.3 Summary

These examples from the road traffic domain illustrate the use of deictic primitives for providing a simple description of surveillance tasks that does not require variables denoting objects. This representation leads to an implementation in HIVIS-WATCHER that does not need as much “state” as HIVIS-MONITOR because we do not need variables to denote all the scene objects in the world. Routines are intended to provide a generic approach that can represent everyday activity. As already mentioned, we cannot describe how routines operate fully here because we need to consider control issues which are discussed in the next chapter. Currently we only have half the story together with a model of what we want the various routines to recognise (e.g., overtaking, giving way and queues).

This use of routines to model everyday behaviour is describe by Agre and Chapman, for example, in PENG [3] there is the routine of pushing an ice cube at an approaching bee, in BLOCKHEAD [33] there is the routine of junking the stack of blocks above the required block, and in the breakfast example from [2] there is the routine of chopping a banana. There are numerous other routines we engage in, such as when drinking coffee, and in HIVIS-WATCHER we are using Agre and Chapman’s approach to represent those routines of the official-observer and those of the scene objects.
8 Conclusions

In this chapter we have described the representation of events and behaviour in HIVIS-MONITOR and HIVIS-WATCHER. This has included two computational theories of how to recognise known behaviour from a stream of intermediate-level visual processing results. These correspond to the script-based and situated approaches we identified in chapter 2. The first computational theory is represented by the HIVIS-MONITOR program which was developed to illustrate the competences of SPATIAL-LAYOUT we described in chapter 3. It began as a testbed for this purpose, providing examples of how SPATIAL-LAYOUT can be used to describe object behaviour in terms of spatial context. The difficulties encountered with extending the data-driven, passive vision approach to provide spatial representation and reasoning that supports more complex multiple object interactions led to the identification of a number of limitations. Addressing these limitations required a reformulation of the surveillance problem and the development of the second computational theory that uses a more task-driven, active vision approach, incorporating deictic representations and reasoning to describe both the scene objects and the official-observer.

The event-detection stages from each HIVIS-system described here illustrate one of their key differences. In HIVIS-MONITOR it is not possible to select which operators to use at run time. In contrast, operator selection is task-dependent in HIVIS-WATCHER so that the number of operators present in the system no longer affects performance. This change to task-directed reasoning is central to supporting local deictic reasoning because of the increase in computational requirements this approach requires.

The main points in this chapter have been:

- The development of HIVIS-MONITOR, which has illustrated how SPATIAL-LAYOUT can be used to obtain contextually relevant information concerning the scene objects. We have also shown that contextual indexing scales well with the number of scene objects.
- The identification that the data-driven approach used in HIVIS-MONITOR, although based in traditional AI techniques, has inherent limitations due to its use of passive vision.
- The use of a dynamic interval tree in HIVIS-MONITOR to store properties attached to temporal intervals.
- The representation of an object's path by using conduits, which alleviates the need for the separate representation of its spatial and temporal descriptions. In our analogical representation, conduits are used to integrate this spatiotemporal description, providing easy access to the data in a uniform way. In HIVIS-MONITOR conduits are used to represent single vehicle histories, which are initially processed separately to allow a coarse grained parallelism, and then combined to enable reasoning about object interactions.
- The reformulation of the surveillance problem from the identification of all scene behaviour for a query system, to the single task surveillance that operates in the "here and now".
- The use of the local frame-of-reference to provide better, more natural, description of the spatial properties of the scene objects.
The identification that processing all scene objects is an important issue, particularly because the cost of processing is dependent on the number of scene objects. We addressed this issue by separating attentive and preattentive operators. To demonstrate this framework we have identified some gestalt primitives that act as simple preattentive operators and can be applied globally over the ground-plane because of their low computational cost.

The objective of this chapter has been to identify how the spatio-temporal primitives can be used to address the surveillance problem. A necessary part of this is event-reasoning (the identification of behavioural primitives), for which we have identified two computational theories. The first, in HIVIS-MONITOR, is data-driven, with event-reasoning following the propagation of visual data from spatial primitives through events and onto episodes in a pipelined way. The second, in HIVIS-WATCHER, is task-driven, with event-reasoning generated as a side-effect by the operators selected as part of the evolving routines that provide perceptual understanding.

The description of event-reasoning in HIVIS-WATCHER is not complete, we do not compose events to provide temporally bounded episodes. This subject has been described in the context of HIVIS-MONITOR and perhaps adopting a data-driven solution that operates on contextually identified events is viable, however, it might compromise the task-driven framework. Further investigation is a subject for future work involving more abstract control (see chapter 5 section 10.3).

Although not as complete in its description, this second version corresponds more closely to the perceptual behaviour of an official-observer engaged in understanding the activity taking place in the environment. Making the computational theory more situated is intended to put emphasis on the visual behaviour performed by the machine based vision system, something that is not clear in HIVIS-MONITOR’s pipelined architecture. We develop this situated approach used in HIVIS-WATCHER in the next chapter.

In the implementation of HIVIS-MONITOR developed here, there are constraints on the range of episodes that can be detected, which we separate into bottom-up and top-down forms. In the bottom-up class are restrictions to the set of aspects due to the available range of common primitives, which in turn are restricted by the range of results produced by intermediate-level visual processing. For example, the intermediate-level visual processing compact-encoding results used here do not include colour information so we cannot have any aspects that represent the colour of scene objects. In the top-down class of task-layer control, restrictions are due to the need for specifying how (and which) events and episodes are detected (see assumption 8), and also from the surveillance problem itself, which constrains the set of episodes that are detected to a subset of those present in the scene (because we are only interested those properties relevant to surveillance). In the next chapter we extend this top-down task-layer control.
In chapter 4 we saw that the lack of situatedness in HIVIS-MONITOR prevented it from fully addressing the surveillance problem. We claim that the incorporation of the situated approach described in chapter 2 enables the perceiver to more fully understand the evolving context of the observed participants in the environment. This is made possible by the introduction of representation and reasoning that is built around the "here-and-now" quality of everyday life, providing a solution to on-line understanding which we propose is the way forward in addressing the surveillance problem. This provides a more natural approach to understanding object behaviour because, as described by Heritage [97, pages 60-61], the task of fellow-actors is necessarily one of inferring from a fragment of the others' conduct and the context, what the others' project is, or is likely to be. We make this task more specific by also incorporating the expectation of the perceiver in the form of a given surveillance task or question.

In chapter 4 we began work on the situated approach by extending the domain- and inference-layers, which provide the foundation to our second implementation called HIVIS-WATCHER. In this chapter we investigate a suitable task- and strategic-layer approach that can use the extended functionality to demonstrate how selective attention can enable a vision system to perform effectively. The high-level vision component developed here uses Bayesian networks combined with a deictic representation to create a dynamic structure to reflect the spatial organisation of the data, and measure task relatedness. Together these give attentional focus making the reasoning relevant to the task.

After we present an overview of HIVIS-WATCHER we investigate the forms of task- and strategic-layer knowledge needed to perform a set of surveillance objectives that increase in complexity under the headings: proximity, coordination and grouping. These illustrate the necessity for increasing amounts of perceiver participation, from selecting which objects to attend, to organising what reasoning should be performed on each object, area of environment, or group of objects. These control mechanisms enable us to illustrate various forms of spatial reasoning that can be used to provide an understanding of the reactive behaviour of the participants in the scene.

---

†In this chapter, sections 1 to 6 are based upon the conference paper "Selective attention in dynamic vision" (Howarth and Buxton [107]). The use of conflict-areas in section 8 is based upon work described in the conference paper "Scheduling in both space and time" (Tsang and Howarth [235]).
1 The perceiver

In this section we describe a system architecture for HIVIS-WATCHER that uses the description of modularity given in chapter 2 section 4.3.5 where we introduced Fodor's separation of input- and central-systems. The input-system we are using consists of the operator processor that we described in chapter 4. Here we describe the central-system. In contrast to the previous two chapters, here we are dealing with representation and reasoning held at the task- and strategic-layers, which are used to control the extraction of the deictic primitives introduced in chapter 4. Surprisingly, instead of giving a straightforward solution, this additional functionality has caused fresh problems that were not present in HIVIS-MONITOR. There are three main problem areas called “functional isolation”, “viewpoint integration”, and “blinkered”. All three problems concerns how data is interpreted in the current context,

- **Functional isolation** (which was introduced in chapter 2 section 3.2.3) is where deictic references do not always refer to the same thing. A deictic reference like the-nearest-object-to-me provides an example, consider the situation where we are walking along the counter in a self-service cafeteria, the-nearest-object-to-me changes as each new object or food item becomes the most proximate and subject to possible selection. In this situation it does not matter that the same deictic reference points to different objects. In other situations the issue of unique identification is important. Consider sitting in a bar trying to work out from a closely packed collection of identical looking drinks which one was mine. In the surveillance problem we do not need the unique identification of each object in the world, instead, we only need to identify those objects that are important to the current surveillance task such that we retain a temporal continuity of identification while each object is important to the official-observer.

- **Viewpoint integration** results from reasoning about multiple scene objects. This concerns how the perceiver reasons about the observed properties of each object that are associated to the object itself and its local environment via the object's frame-of-reference. This causes a data-association-problem\(^1\) that can be resolved by the fact that the perceiver knows to which scene objects it is attending. This problem does not arise in the work of Agre and Chapman because they only have one main character that the system controls, and they are not concerned with understanding the behaviour of the other participants since it is not necessary in their application domains.

- **Blinkered** refers to not perceiving everything in the environment. Often when we interact with the world it is with some purpose in mind. Surveillance usually has a purpose and this is reflected in both how the observation process is performed and the result of the observation. The same scenario, seen by different observers with different tasks, may yield different results because, what is interesting to one may not be of interest to another. This purposive approach enables us to grade the available features and select the essential properties from the various sensory inputs. Without such selectivity we would

\(^1\)The problem of binding related data together.
be overwhelmed by the available data. The benefit of using this task bias makes the reasoning performed relevant to the task at hand. This blinkering problem means that HIVIS-WATCHER is not equivalent to HIVIS-MONITOR (where the perceiver observers everything) because it can miss things.

### 1.1 Basic Architecture

To address these problems the HIVIS-WATCHER architecture has three main parts, called, the "VIRTUAL-WORLD", the "PERIPHERAL-SYSTEM" and the "CENTRAL-SYSTEM". The last two draw their names from the input/central split. The VIRTUAL-WORLD has its name because it contains a model of the static and dynamic environment. The dynamic data is provided by the stream of compact encodings, which are stored so that each frame update is only accessed by operators contained in the PERIPHERAL-SYSTEM. These are selected, given arguments by, and return results to, the CENTRAL-SYSTEM.

The interaction between these three parts is determined by a clock that ticks at every new frame-update. On the arrival of a new frame, the virtual-world is updated, with the new frame of compact-encodings replacing the old frame. Following this update, the selected operators in the PERIPHERAL-SYSTEM are allowed to run to completion, presenting their results as input to the CENTRAL-SYSTEM. The CENTRAL-SYSTEM has inputs and outputs and gates in between, and when the inputs change, the CENTRAL-SYSTEM is run. When running the CENTRAL-SYSTEM, the ready lines on the outputs need to wait until the circuitry has all settled down before admitting the outputs into the PERIPHERAL-SYSTEM. The output from the CENTRAL-SYSTEM represents effector instructions saying which operators to run on the next clock tick when we are given a new frame update. HIVIS-WATCHER only has one clock which is driven by the frames from the intermediate-vision component. The PERIPHERAL-SYSTEM that contains the operators, and the CENTRAL-SYSTEM that controls the behaviour of the system do not need a clock.

This organisation is similar to that described by Chapman [34, page 173] and is summarised in figure 5.1(a). The interaction between the PERIPHERAL- and CENTRAL-SYSTEMS corresponds to the situation where, given the configuration of the materials to hand, it is

---

| (1) | Update buffer with new frame of compact-encodings. |
| (2) | Run peripheral-system; present inputs to the central-system. |
| (3) | Run the central-system. |
| (4) | Run the effector-system. |

[Figure 5.1: Top level loop.](#)
Figure 5.2: The process cycle. The virtual-world holds the application domain-layer data. The peripheral-system holds the inference-layer data. The central-system holds the task- and strategic-layer data.

obvious what to do next. Once you have done this, the next thing to do is likely to be obvious as well. From this, complex sequences of actions result, that have not required a complex control structure to decide what should be done. Some of these sequences of action correspond to routines, but they do not need to. This architecture supports part of the situatedness we want to include in our HIVIS-based systems, allowing us to model routine activity, and because of this it has been adopted in HIVIS-WATCHER. In Agre and Chapman’s implementations the system is only used to reason about how one actor interacts with the environment, in HIVIS-WATCHER we want to address the problem of reasoning about multiple objects, to do this we have extended the basic architecture.

1.2 EXTENDING THE CENTRAL-SYSTEM

To perform multiple object reasoning in a task oriented way we extend the central-system to include “local-” and “global-forms”. The local-form uses the observer’s interpretation of an object’s deictic frame-of-reference. For each attended to object, which we call an “agent”, the perceiver accesses environmental properties that are local to the agent. The local-form represents what the perceiver believes the agent is doing, this is unlikely to be the same as what the object believes (if it has beliefs) although the assumption is that some similarity should hold. Consider the difference between a spectator at a sports-event and the athletes taking part, this difference holds in other everyday situations as well. In HIVIS-WATCHER we take the role of a spectator, with the central-system representing what the perceiver knows of the environment not what any other agent in the world knows.

The global-form represents the observer’s “whole scene” viewpoint and is used to: “allocate” agents, “collect” together their results and provide an integrated description of what is happening in the scene. This extension is summarised in figure 5.1(b) and in figure 5.2.
This completes our top-level tour of HIVIS-WATCHER. We have described how the architecture as a whole provides a framework for addressing the functional isolation problem, how the collect component provides the main solution to the viewpoint integration problem, and how the task-layer in global-form turns the blinkered problem into a design feature.

In section 2 we describe the contents of the virtual-world, which is connected to the central-system via the peripheral-system. We described the peripheral-system in chapter 4 section 6.3 which covered the operator processor, with the operators themselves discussed in other sections of that chapter.

In section 3 we begin our description of the central-system with the operation of the agents in the local-form. Then in section 4 we describe the organisation of allocate and collect in the global-form. These elements of HIVIS-WATCHER are illustrated in figure 5.2 which also shows the tight coupling between the peripheral- and central-systems providing the foundation for reasoning in the “here-and-now” with no backtracking and no reasoning in hindsight. This describes the computational approach and in section 5 we present details of how it has been implemented.

In section 6 we demonstrate how these parts of the HIVIS-WATCHER program operate using examples taken from the road-traffic domain. Although these examples are domain specific the intention is that the techniques developed here can be adapted to other application domains. The results from these examples highlight some limitations that we address in section 7 where we extend the computational approach with the kernel. After providing implementation details we demonstrate how this addition solves the identified limitations. Two example are given in sections 8 and 9 with the second one opening up some areas for future work. This completes the description of HIVIS-WATCHER.
2 Virtual world

The virtual-world acts as the interface with the real world, with the data held in this part of HIVIS-WATCHER forming a virtual environment that the rest of HIVIS-WATCHER uses. This interface is common to both HIVIS-based systems and the reason for its presence is given in chapter 1. The virtual-world is not an internal "mental" representation of the world, instead it is a representation of both static a priori knowledge and dynamic data from intermediate-level vision processing. The virtual world represents the signal from a vision component whose global "overhead" viewpoint has been obtained from a fixed camera position. The virtual world representation does no runtime spatial reasoning, it is used solely to hold information for access by the peripheral-system. If HIVIS-WATCHER had direct links to the vision system then the visual operators would directly access the available vision system results. Instead we are using a time frame stamped encoding from the visual processing. As shown in figure 5.3 the virtual-world of HIVIS-WATCHER holds four sub-components called the "buffer", "spatial-layout", "traffic-flow-model", and "effector-output".

2.1 Buffer

The buffer holds the dynamic perceptual data from the intermediate-vision component that arrives as a frame of compact encodings (i.e., 3D pose positions). The dynamic data provides the domain-layer knowledge about the scene objects and is replaced on each system clock tick when a new frame of compact encodings is given. The buffer holds a list of buffer slots each of which can be allocated to a marker by giving the marker the name of the buffer slot. The correspondence between a buffer slot and a scene object's frame update is made possible because of the name given to it by the intermediate-vision component. Buffer slots that are not given a new frame-update are removed. A buffer slot is assigned to each scene object as it appears, with the appearance being triggered by the need for the buffer to create a new buffer slot. The set of current buffer slots represent the set of markable objects. Markers access the buffer via a descriptor to the buffer slot that holds the relevant data structure. Tracking is achieved by the updating of each buffer slot. We distinguish between markers that point to a buffer slot via a descriptor and the name of the descriptor which we call an "index". Indexes are used for preattentive processing, and markers are used for attentive processing.

2.2 Perceived environment

The perceived environment mainly consists of the spatial-layout which we described in chapter 3. However, it does not fully comply with the stated aims of the virtual world because the separation between environment and internal representation is not complete. The domain-layer information also includes supportive semantic data that is internal to the perceiver. It would be possible to separate these and hold them in two distinct databases, but this has not been done. A less important role is played by the traffic-flow-model.
Figure 5.3: An overview of HIVIS-WATCHER showing the elements that make up the virtual world.

The virtual-world draws upon the analogy between the real world optic-array and storage representations that mimic this structure. For example, in Chapman's program "BLOCKHEAD" [33] the real world just contains blocks, each having an identifying property (i.e., a distinguishing colour or alphabetical letter) on its visible face. The virtual-world is a 2D array, the contents of which denote each block's position by holding its identifying property at an array address that denotes its position in the world. The table top that supports the blocks that rest on it, is represented by the bottom row of the array, each address of which holds the value denoting "table". Free-space is denoted by the addresses that do not have a value. A marker pointing at a block, just holds the coordinates of its array address. Effectors that change the real world, just change the contents of the 2D array. This example illustrates the simplicity of the representation of the perceived world, however, this is sufficient to support all the reasoning used in BLOCKHEAD.

The traffic-flow-model holds a global view of all recent vehicle positions and maintains a model describing the average paths taken by scene vehicles. The representation of dynamic information about the flow of traffic would be useful, but to do this properly would involve some form of learning. This is a subject for future work and is briefly discussed in appendix C.

2.3 Display

An important element of the virtual world is to display the contents of the buffer and various forms of marker attachment to the user. The effector-output also holds the log of events detected by HIVIS-WATCHER which represents the answer to the user's question.
3 Scene agents

An instance of the surveillance problem is identifying visitors by their unusual behaviour as they walk through the office space. To do this requires having a model of what the typical everyday behaviour would be. This is used to both focus the reasoning and highlight unusual actions. People who know an office well walk confidently about, taking short cuts, etc. Visitors on the other hand, spend time looking around rooms, use longer routes, look at door numbers, ..., and tend to stray from the typically used paths. In this section we describe a representation of single object typical behaviour called "typical-object-model", we show how a model written using this representation can be "fitted" to an observed scene object, and then we extend this approach to multiple scene objects.

3.1 Typical object model

Scene agent behaviour is represented by a collections of rules called the "typical-object-model" (TOM) which is part of the central system, defining how HIVIS-WATCHER interprets scene object activity. Agre and Chapman's work (e.g., their program "PENGI" [3]) is the main design influence upon the TOM. Agre [2] and Chapman [34] seem to have addressed the problem of how to provide system behaviour that fulfills the situated approach, and to claim that they provide reflexive accountability2. This is important because, as described in chapter 2 section 3.2.4, the whole idea of using context-free rules is flawed. The implementation of MACNET (see appendix E) was a first step towards a situated TOM for the traffic surveillance application.

The scene objects we are interested in all have an intrinsic front, allowing the actions of such objects to be described in terms of their own local coordinate system (see chapter 4 section 6). We access the properties of these objects via the peripheral system. During each clock cycle, the selected scene objects, called "agents", each have the TOM run on them. The main purpose of the TOM is to identify events and select which operators in the peripheral system are to be used on the next clock tick. This enables the system to obtain information that is pertinent to the attended object. The results from the selected operators are fed back to the appropriate agent providing task related information upon which to base it's next selection. The agent's deictic viewpoint helps to localise spatial and temporal reasoning, making it agent specific. The central premise behind this agent selective viewpoint is that an agent need not name and describe every object in the domain, but instead should register information only about objects that are relevant to the task at hand. At any one moment the agent's internal representation should register only the features of a few key objects and ignore the rest.

As shown in figure 5.4 the inputs to the TOM, called "aspects", are from the operators run by the peripheral system on the current clock tick, and the outputs are called "effector commands". Although the TOM performs no computation on domain-level data

2Reflexive accountability was described in chapter 2 section 4.2 in definition 2.2.
itself, it influences what inference-level computation is performed by the peripheral system.

The rules in the TOM, called "arbiters" in figure 5.4, are defined off-line before runtime and fixed during execution. The set of rules, $R$, represents the action of a prototypical object to a given situation, $S$, and recent history, $H$, (internal state of one time step). These rules express relationships between observed situations and effector commands. These effector commands are used to describe both detected events, $E$, and which operators, $O$, to use on the next clock tick. This can be summarised by: $R : S \times H \rightarrow E \times O$. This shows the correspondence to the "behavioural component" described by Whitehead and Ballard [252]. MACNET is compiled to produce a circuit of combinatorial logic (e.g., andg, org and invert gates for $\land$, $\lor$, and $\neg$ respectively) which is extended to include other gates such as latch. The latch gate is able to remember the value given to it on the previous clock tick, and its presence makes the TOM’s circuit context dependent, providing the recent history for $H$. Any language that provides the functionality of $R$ could be used instead of MACNET however, as shown in table 5.1, such a language also needs to be able to pass and select arguments to the operator that the rule guards.

The implementation of the TOM in MACNET is in two separate parts. The first embodies the rules of how agent activity is understood by the perceiver. The second is the data-structure that holds the data about each currently selected agent. The mechanism that supports this is called the "AGENCY" and is described below.

3.2 AGENCY

The AGENCY holds one copy of the static circuit of the TOM, together with a dynamic runtime environment (DRE) for each agent. Each DRE holds the value-, latch- and result-arrays that describe the runtime context of the agent (see appendix E section 2). The latch-array provides the values from the previous clock-tick and is given a value for the next clock-tick providing an evolving and separate context for the evaluation of each agent’s TOM circuit.

The system is currently set up with just three agents. After the AGENCY has run each agent in its own separate evaluation environment the results from each agent’s local view
Each MACNET rule is made up of a number of components that are linked together by the rule name $\mathcal{R}$. Each rule has just one arbiter.

- **propose-default** – the basic set of values that are to be given to $\mathcal{R}$’s arbiter.
- **propose** – these override defaults and are the regular proposals that are given to $\mathcal{R}$’s arbiter. Each proposal, $P_i$, has at least one condition $C_i$, and a set of arguments $A_i$. There can be zero or more proposals for a rule (i.e., $0 \leq i$).
- **condition** – defines the things that need to be true before we even look at the proposal $P_i$. The condition $C_i$ represents. For each $i^{th}$ proposal there can be one or more conditions (i.e., $1 \leq j$), usually one is enough.
- **override-proposer** – when one proposal should be preferred to another.
- **arbiter** – provides an effector-command as output which is given the arguments $A_i$, when the condition $C_i$ of proposal $P_i$ is true. The effector-command is typically for a peripheral-system operator.
- **registrar** – collects the inputs given by the peripheral-system.

Table 5.1: The main MACNET language features.

of activity are integrated by the global-form in the central-system which determines what may be happening in global terms, and which we describe in section 4. The benefit of running the same static circuit for each agent over having three copies of the same circuit is that it reduces storage.

### 3.3 Example

The rule set is domain-layer knowledge and currently in the road-traffic domain implementation of HIVIS-WATCHER we just consider one road-vehicle type. Trams, for example, would require a different model of dynamic action. Each operator that is used by the TOM has a rule defined in the MACNET language, a summary of which is given in table 5.1. This provides a correspondence between rules and operators that crosses the clock-tick boundary (i.e., the rules select the operators on the data from time $t$ and the selected operators will be run on the new data that arrives at $t+1$). This can be annoying as the effects from what is seen now is not done now, instead it is performed one time step later.

The use of MACNET to express routines requires setting enabling flags in the conditions that permit the next stage to fire on the next clock tick. The enabling flag provides a simple form of evolving context that prevents their activation when they are not contextually relevant. Here is an example, taken from TOM used in the road-traffic domain.

```prolog
(arbiter finds-speed-difference? (object doit?) (propose-default ignore :object (constant *f*) :doit? (constant *f*)) (propose findout :object *the-other* :doit? (constant *t*)) (condition findout (andg (eqm *the-other-has-location* (constant :back)) ;; behind-me and (eqm *the-other-has-speed* (constant *f*))) ;; no previous call.
```

The TOM is also used to identify events, such as when the positional aspect the-other-is-behind becomes true:
Table 5.2: The set of arbiters in the TOM that generate effector requests.

<table>
<thead>
<tr>
<th>event arbiters</th>
<th>operator arbiters</th>
<th>abstract arbiters</th>
</tr>
</thead>
<tbody>
<tr>
<td>event-behind-me</td>
<td>find-velocity?</td>
<td>refobj-slowing-down</td>
</tr>
<tr>
<td>event-beside-me</td>
<td>find-orientation?</td>
<td></td>
</tr>
<tr>
<td>event-infront-of-me</td>
<td>find-occupiedregions?</td>
<td></td>
</tr>
<tr>
<td>turning-event</td>
<td>find-nearest?</td>
<td></td>
</tr>
<tr>
<td>in-turning-region-event</td>
<td>find-orientation-change?</td>
<td></td>
</tr>
<tr>
<td>heading-event</td>
<td>publicize-stationary-object!</td>
<td>change-motion-prior!</td>
</tr>
<tr>
<td></td>
<td>find-speed-difference?</td>
<td>find-heading-difference?</td>
</tr>
</tbody>
</table>

(arbiter event-behind-me (event-value event-type doit?)
  (propose something-behind-me :event-value *other*
   :event-type (constant :behind-me) :doit? *the-other-is-behind*))

And region occupancy events such as being in a turning region:

(arbiter in-turning-region-event (event-type doit?)
  (propose-default nothing :event-type (constant *f*) :doit? (constant *f*))
  (propose enter-turning-region :event-type (constant :enter-turning-region)
   :doit? (constant *t*))
  (condition enter-turning-region (andg *turning* (invert (latch *turning*))))
  (propose inside-turning-region :event-type (constant :inside-turning-region)
   :doit? (constant *t*))
  (condition inside-turning-region (andg *turning* (latch *turning*)))
  (propose exit-turning-region :event-type (constant :exit-turning-region)
   :doit? (constant *t*))
  (condition exit-turning-region (andg (invert *turning*) (latch *turning*)))
  (propose booting :event-type (constant *f*) :doit? (constant *f*))
  (condition booting (latch *booting*))
  (override-proposer booting exit-turning-region inside-turning-region
   enter-turning-region))

The results from the event arbiters listed in table 5.2 are used to generate the events for each object to which a TOM is attached. The results from the operator arbiters listed in table 5.2 are used to determine if the operator should be run and on what arguments.

3.4 Summary

We have described how individual local agents can be represented by a TOM. Representing how to understand activity in a social context is a difficult problem, which the TOM and MACNET only begin to address. Having developed a TOM that captures the observer’s notion of what the object is doing it seems natural to extend this to include what the agent can perceive. However, such an extension is unlikely to be of much use because we would be dealing with data that is not be available to the official-observer such as the occluded faces of the other objects.
4 Task based control

Task based control is used to reduce the computation performed by resolving two key problems encountered when processing a stream of frame updates, which are: (1) the computational load of determining what all the objects are doing, for instance, some tasks may not require considering all object interactions; (2) the amount of evidence (see chapter 2 section 4.1) which involves managing the temporally evolving mass of input data, an issue even with the relatively compact representation provided by the set of pose positions. We also address the problem of viewpoint integration described in section 1.

HIVIS-WATCHER represents a stationary observer watching a scene that contains a number of moving objects in its field of view. The observer is looking for "interesting" interactions between two or three objects that fulfill some measure of how similar the interaction is to an instance from the perceiver's set of known behaviours. These interactions may take a number of image frames, and the interaction might complete or be initiated out of shot. The typical-object-model in the local-form (see figure 5.2) itself is not really capable of reasoning about events and actions, it can only detect them, because it is too dumb and too reactive. Also the typical-object-model run by each agent operates at the wrong computational level to act as an attentional controller for the official-observer.

To enhance these features so that the central-system can do event reasoning, the global-form has been added. The global-form is where all the relevant agent results are collected together, producing a single consistent story capturing those features of the scene's happenings that were deemed interesting.

A fundamental step in the approach taken here is to separate the simple and complex operations that act upon the input data, and use the results from the simple operations to guide the application of appropriate complex functions. The goal here is to reduce the irrelevant application of the more computationally expensive complex operations by ensuring the reasoning performed is relevant to the task at hand. In the context of understanding what a scene object is doing, simple operations include those peripheral, preattentive cues for position, velocity, nearest-object, while complex operations include attentional, agent based computations such as path-prediction and other-object-location. The connection with perception is when we consider the simple operation to be a peripheral one (e.g., motion detection) and the complex one to be a foveal one (e.g., recognition). We introduced previous work on this in chapter 2 section 4.2, where we described foveated attention and also described Pylyshyn's FINST approach [191, 190] for dealing with multiple foveated objects. We can summarise this by considering a general case:

**Definition 5.1** Given two operators OP1 and OP2 where (1) OP1 is much less complex and runs faster than OP2, (2) OP1 does not return an answer that is as useful as OP2, (3) OP1 is true in all situations where OP2 would provide a useful result (4) OP1 is false in all situations where the preconditions of OP2 would not be met or where we do not want to apply OP2. If these conditions are fulfilled, OP1 can act as a guide for the application of OP2. We call OP1 the simple operator and OP2 the complex operator.
The **global-form** acts like a switch allocating and receiving input/output from the agents it is running. In practice the general case is more complicated because the selection of which **OP2s** to run is dependent upon factors other than the **OP1** results alone.

### 4.1 Guiding Computation

The traditional passive approach used in **HIVIS-MONITOR** only allowed the user to find out what had happened by querying the database created after the observation data had been processed. In contrast, in the situated approach the user asks the question before **HIVIS-WATCHER** begins looking for anything. The question takes the form of the tuple, called a “policy” or surveillance task, \((\text{cue}, \text{attend}, \text{ignore})\) which specifies the simple operator **OP1** that acts as a preattentive cue, a set of behaviours to look for, and a set of behaviours to ignore. The policy defines those features that are interesting. Each policy has a primitive “cue” which may be the same for more than one policy and may apply to policies other than those selected. The policies considered in this chapter concern the identification of all occurrences of some behaviour such as “closing the door”. This causes **HIVIS-WATCHER** to be blinkered to any observed behaviour that is not related to the current policy.

The policy controls the allocation of agents. Some examples of simple policies include: (1) describe the path of just one (or a selected number) of vehicles; (2) “track all new objects” which would cause **HIVIS-WATCHER** to either assign an unassigned marker to the new object, or move a marker from a currently marked vehicle to the new one. These examples are simple because they do not depend on the behaviour exhibited by the scene objects. In this chapter we investigate more complex polices such as “identify likely overtaking behaviour”.

### 4.2 Allocate

The allocation process assigns an agent to a scene object so that the typical-object-model can generate events and select operators related to this object’s behaviour. The **allocate** component provides a solution to the **computational load problem** introduced at the beginning of this section by identifying those objects that the policy cue deems to be the most interesting. This attentional system has three properties that are used to guide the reasoning of the runtime system. (1) **Focus-of-attention** – only the selected set of target hypotheses and their associated functions are updated. (2) **Terminate-attention** – once a target hypothesis has been confirmed or denied to an acceptable degree of confidence we can stop all activities associated with it. (3) **Selective-attention** – we can dynamically select the current most interesting hypothesis to watch.

The objective of the **allocate** component is to direct attention. Terminate attention is determined from data given by **collect**, which is used to decide what to do when a situation is identified (a task goal achieved). Fast identification of uninteresting policy is important, so that more time can be spent looking for relevant features. The **allocate** component is used to decide whether **HIVIS-WATCHER** should ignore the objects involved
in the exhibited behaviour or continue to attend to them. This allocation process is task
driven not data driven. By allocating agents in this way we only process data that is
related to the task, with the processing performed also made context dependent via the
typical-object-model.

4.3 Collect

The collect component collects together the results from the agents. Each agent provides
a sequences of activities which collect converts into an episode. This episode takes into
account everything (or at least the important features) that have been detected from the
recent frame updates. The collect component measures the typical-object-model results
against the “happenings” in the scene to determine whether the real action is legal (typical,
untypical) or illegal. This contextual knowledge is used to enable HIVIS-WATCHER’s
global-form to make sound predictions about what is happening in the scene. The
global-form operates with a longer time frame than the agents in the local-form and
is able to combine the various events detected by the agents into a continuous story of
what is happening in the scene. This enables HIVIS-WATCHER to cope with situations
where no clear behaviour is evident.

The design of the collect component solves the viewpoint integration problem caused
by the use of deictic representation. This problem is simplified by the fact that the official-
observer originally selected which objects to attend and thus the execution of the typical-
object-model is like the official-observer considering the possible next actions of an object
it is attending. The collect component provides a solution to the viewpoint integration
problem and reduces the amount of evidence that needs to be retained to task related
values.

4.4 Summary

The task based control described here addresses the problems of computational load,
amount of evidence, and viewpoint integration. The solution is at the cost of complete
observation (i.e., the blinkered problem is part of the solution). In chapter 2 section 4.1
we introduced a lattice representing all possible surveillance tasks. The top of this latticce is equivalent to complete observation and the various subsets correspond to different
combinations of the possible surveillance tasks, including no surveillance task. This model
is “ideal” because tasks affect one another due to conflict of process or availability of
operators, and also because it may not be possible to define the set of all possible tasks.
We claim that complete observation is not needed for surveillance, and demonstrate the
advantage of this approach for ignoring irrelevant details, and the drawback of missing
what you do not look at. Guiding what is attended to in the field-of-view, in this way, is a
major difference from HIVIS-MONITOR, and involves guiding the reasoning of the whole
vision system, acting as an attentional mechanism that exploits knowledge of the specific
problem being solved.
5 Implementation and Formalism Details

Section 4 provided the computational approach to task based control that is used in HIVIS-WATCHER. This section presents details of the position of allocate and collect in the HIVIS-WATCHER framework, and two applications of Bayesian networks that are used in this implementation. We are using Bayesian networks as a standard AI technique for representing and reasoning about uncertainty (see appendix D for some background to the mathematical foundations that Bayesian networks provide). We do not extend the computational theory of Bayesian networks, although our first application does present a novel use of them, called "Dynamic Decision Networks" (DDN). This uses dynamic networks that are similar to Bayesian networks in a dynamic context, with the graph structure updated to reflect the contents of the scene as represented by the current frame of compact encoding. To express these manipulations of the graph structure we develop a concise (and intuitive) language called "MACDDN".

5.1 Framework

To place allocate and collect in context we first provide a more detailed description of the functional components in HIVIS-WATCHER. Table 5.3 describes the main data elements that are the arguments to the functions, which are given in order of application. In table 5.3 and its related figure 5.5 the general mnemonics are as follows: $W$ is the world, $P$ is the perceptual operators, $B$ is behavioural, $M$ is sensory-motors. In figure 5.5 the arrows are the functions and the boxes are the data elements. The dots on the function arcs denote two cases: when an arrow function has more than one argument, and when an arrow function sends a copy of its result to different destinations. This figure provides a more detailed organisation of the different components that we have been discussing. In figure 5.5 the ordering of the functions expands on that illustrated in figure 5.2, with the order of execution corresponding to the top level loop given in figure 5.1. In the functional description: allocate corresponds to the two functions for updating the DDN (called $B1$) and allocating agents ($B2$); and collect corresponds to the function for collecting results ($B4$) and part of the function for selecting effectors ($M1$).

5.2 Dynamic Decision Networks

A dynamic decision network (DDN) has the objective of modelling data that is supplied incrementally over time in discrete steps or clock ticks. It cannot model continuous time because the process of graph extension in necessarily discrete. The graph structure changes overtime to reflect the temporally evolving properties it represents. To do this we run the Network Expansion and Inference (NEI) algorithm given in figure 5.6. If the graph structure is constant over time a better solution would be the more usual static graph approach (for details see Pearl [183]), because using a DDN incurs the additional overheads of the update-graph-structure step. We can distinguish between the different form of Bayesian network in two distinct ways: topological structure (either continuous or
Figure 5.5: Functional description.

### Data

<table>
<thead>
<tr>
<th>description</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>world</td>
<td>W</td>
</tr>
<tr>
<td>the buffer</td>
<td>B</td>
</tr>
<tr>
<td>spatial layout</td>
<td>SL</td>
</tr>
<tr>
<td>effector arguments</td>
<td>E1</td>
</tr>
<tr>
<td>effector args for DDN</td>
<td>E2</td>
</tr>
<tr>
<td>basic aspects</td>
<td>BA</td>
</tr>
<tr>
<td>basic for DDN</td>
<td>B1</td>
</tr>
<tr>
<td>basic for agents</td>
<td>B2</td>
</tr>
<tr>
<td>elected aspects</td>
<td>SA</td>
</tr>
<tr>
<td>collected results</td>
<td>R</td>
</tr>
<tr>
<td>agents for markers</td>
<td>AL</td>
</tr>
<tr>
<td>agency input</td>
<td>I</td>
</tr>
<tr>
<td>agency output</td>
<td>O</td>
</tr>
<tr>
<td>display</td>
<td>D</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>order</th>
<th>description</th>
<th>mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>frame update</td>
<td>( W : W \rightarrow B )</td>
</tr>
<tr>
<td>2</td>
<td>basic operators</td>
<td>( P1 : B \rightarrow BA )</td>
</tr>
<tr>
<td></td>
<td>selected operators</td>
<td>( P2 : E \times P \rightarrow S )</td>
</tr>
<tr>
<td></td>
<td>copy basic</td>
<td>( B0 : BA \rightarrow B1 \times B2 )</td>
</tr>
<tr>
<td></td>
<td>update DDN</td>
<td>( B1 : B1 \times E2 \rightarrow AL )</td>
</tr>
<tr>
<td>3.1</td>
<td>allocate agents</td>
<td>( B2 : B2 \times SA \times AL \rightarrow I_i )</td>
</tr>
<tr>
<td>3.2</td>
<td>run agents</td>
<td>( B3 : I_i \rightarrow O_i )</td>
</tr>
<tr>
<td>3.3</td>
<td>collect results</td>
<td>( B4 : O_i \rightarrow R )</td>
</tr>
<tr>
<td>4</td>
<td>select effectors</td>
<td>( M1 : R \rightarrow E1 \times E2 \times E3 )</td>
</tr>
<tr>
<td></td>
<td>run display</td>
<td>( M2 : B \times E3 \rightarrow D )</td>
</tr>
</tbody>
</table>

Table 5.3: Functional description using the top level loop shown in figure 5.1.

Figure 5.6: The Network Expansion and Inference (NEI) algorithm.

Figure 5.7: The time-cell as a bucket holding a set of nodes.
changing), and graph structure (one of: causal tree, causal polytree or multiply connected). The Bayesian network described here is a topologically changing causal polytree.

The inputs to DDN are taken from BA (basic-aspects in figure 5.3) outputs. These BA outputs are given once every system-cycle, which means that the values held at time $t_{nov}$ are from the current outputs. At the end of each system-cycle, the storage is "moved" (by pointer reallocation) so that what was held at $t_i$ is now held at $t_{i-1}$. When the peripheral-aspects become available, they are used to re-construct the DDN via update-graph-structure.

### 5.2.1 Connection rules

We constrain the set of possible graph structures by defining a set of "connection rules". These connection rules are very local and are needed to ensure that loops are not formed within the graph which would invalidate the inference algorithm for a causal polytree. The structure of the graph is defined by the set of connection rules that determine the validity of each edge. The connection rules consist of two forms "now-connections" (nowc) and "one-step-temporal-connections" (ostc). The now-connections link nodes that share the current time value, and one-step-temporal-connections link nodes with adjacent time values.

Each node is defined by a tuple $\langle n, o, a, b \rangle$ where $n$ is a node type symbol, $o$ is an owner symbol, $a$ is an attribute symbol, and $b$ is a belief symbol. Although time is important, it is not expressed explicitly as part of each node, instead it is implicit in the node's position within the graph. Time is modelled as subgraphs, called "buckets", that hold all the nodes for a particular time-cell. Figure 5.7 illustrates this idea, with one-step-temporal-connections defining the links that can be made between nodes in adjacent buckets.

For graph construction we use a more compact notation that rewrites $\langle n, o, a, b \rangle$ as $n^o_{a}$ in the case where $a$ is defined, and $n^o_{a}$ if not, and where $a$ acts as a wild card. This notation hides the belief value because it is not used during graph construction, and makes explicit the type, owner and attribute symbols that the construction rules operate upon. This part of the language is used to express the vertices present in the DDN, using the syntax: NT is the set of node type symbols, OV1 is the set of owner symbols, OV2 is the powerset of OV1, VT is the set of node tuples, and OV1$^{\bullet}$ is the set of owner symbols extended by the wild card, i.e., $OV1^{\bullet} = OV1 \cup \{\bullet\}$. In each node tuple $n^o_{a} \in VT$, we have $n \in NT, o \in OV2$, and $a \in OV1^{\bullet}$. These nodes are held in the DDN graph, and we use DDNG to denote the set of DDN graphs, and for which we provide a more precise description in section 5.2.2.

We now describe the three predicate tests we require: nowc(p) says whether the node p is present in the bucket for $t_{now}$; nowc($p \land q$) says whether the both nodes $p$ and $q$ are present in the bucket for $t_{nov}$; and ostc(p, q) says whether the nodes $p$ and $q$ are in the consecutive buckets under consideration. Also, in the definition of these three predicates we will use functions that can inspect a node tuple to obtain the value of a particular slot,
for example, the function node-owner of type VT → 0V2 is used to obtain the owner slot of a node tuple. The syntax of the three predicates is as follows:

- if p ∈ VT then nowc(p) is a predicate
- if p, q ∈ VT then nowc(p ∧ q) is a predicate
- if p, q ∈ VT then ostc(p, q) is a predicate

The semantics describes an interpretation as a tuple \((VW, H, M_{VT})\) where \(VW\) is a universe of node tuples, \(M_{VT}\) is a meaning function with \(M_{VT} : H → VT → 2^{VW_X VW_X H}\), \(H\) is a finite set of deictic time-cell indexes. In the following for any p, q ∈ VT and \(t_{now}, t_1, t_{t-1} ∈ H\). A variable assignment is a function \(VAL : VT → VW\).

- \(VAL[\text{nowc}(p)]\) if \(VAL(p) ∈ M_{VT}(t_{now}, p)\)
- \(VAL[\text{nowc}(p ∧ q)]\) if \(VAL(p) ∈ M_{VT}(t_{now}, p)\) and \(VAL(q) ∈ M_{VT}(t_{now}, q)\)
- \(VAL[\text{ostc}(p, q)]\) if \(VAL(p) ∈ M_{VT}(t_{t-1}, p)\) and \(VAL(q) ∈ M_{VT}(t_1, q)\) and \(\text{node-owner}(p) = \text{node-owner}(q)\)

This semantics for the connection rules is made less concrete by the use of deictic temporal references, because they act like global variables. The \(t_{now}\) time-cell is easily resolved in practice, however, \(t_1\) and \(t_{t-1}\) range over all the time-cells modelled by the current graph with the predicate acting as a guard for the pairs where the test is true.

Buckets are held in an indexable data structure, so that each bucket has a bucket-index \(bi ∈ BI\), where \(BI\) is the set of bucket-indexes. The mapping between time-cells and buckets is held by TS which is an ordered list of \(B\) bucket-indexes, such that each place in the list maps a bucket-index to an elements of \(H\), i.e., \([t_{now}, t_{now-1}, t_{now}, t_{now+1}, \ldots, t_{now+\text{m}}]\) where TS is a \(B\) length list of \(bi ∈ BI\), \(B\) is for an odd number, and \(m = \left\lceil \frac{3}{2} \right\rceil\). (In the implementation we restructured this list to make the \(t_{now}\) clearer by having it at the front of the list, so that the second pointer is to the bucket for the next time-cell \(t_{now+1}\), and the last pointer is to the bucket for the previous time-cell \(t_{now-1}\), which rearranges the positions in this list of bucket addresses to represent the following time-cells \([t_{now}, t_{now+1}, \ldots, t_{now+m}; t_{now-1}, \ldots, t_{now-1}]\).)

### 5.2.2 Construction Algorithm

To form the DDN graph from the set of valid node tuples we use the two forms of connection rule, which determine when a directed edge can be added between two nodes. The connection rules associated with the DDN are task dependent and act on task specific data. Rather than define these we will describe a more general DDN macro language, called "MACDDN", and provide task specific examples when we consider the surveillance tasks in sections 6 and 8.

In the MACDDN language we use the following notation. To add an edge we use the infix operator \(→\) called "add directed edge", that is a function \((→)\) of type \(VT → VT → DDNG → DDNG\), that updates the DDN graph. To make the notation less cluttered we will hide the DDNG graph parameter so that we only need to specify first two arguments of \((→)\).
For example, we will write $p \rightarrow q$ to add an edge between the node tuples $p$ and $q$ where $p, q \in VT$.

In addition to adding edges, the MACDDN language also needs to add a new node. This is done using two functions: make-node(NODE-TYPE, NAME) which creates a node tuple with a wild card attribute (and is a function make-node of type $\text{NT} \rightarrow \text{OV2} \rightarrow \text{VT}$); and add-vertex(DDNG, NODE) which adds a node tuple (for now we will hide the DDNG parameter and define it as add-vertex($p$), where $p \in VT$).

To describe the constructor macros used in the MACDDN language we will use templates that are written using the notation introduced above, together with additional syntax for tests (if antecedent do consequent od) and value assignment (denoted by the := symbol).

- **now-link($p, q$) = if nowc($p \land q$) do $p \rightarrow q$ od**
  which adds a directed edge between two node tuples $p$ and $q$ when nowc($p \land q$) is true.

- **now-ref($p$) = if nowc($p$) do $c := \text{node-owner}(p); \ r := \text{make-node}(R,o); \ add-vertex(r); \ p \rightarrow r$ od**
  this adds a result node to the graph so that the output from the belief slot can be translated to a more useful form.

- **now-relationship($p, q$) = if nowc($p \land q$) do $o := \{\text{node-owner}(p), \text{node-owner}(q)\}; \ r := \text{make-node}(R,o); \ add-vertex(r); \ p \rightarrow r; \ q \rightarrow r$ od**
  this combines the results from two nodes denoting some relationship between them, and is used as part of the solution to the spatial arrangements problem.

- **temporal-change($p, q$) = if ostc($p,q$) do $p \rightarrow q$ od**
  this temporally links nodes that are of different types, but the same owner.

- **temporal-continuity($p$) = if ostc($p,p$) do $p \rightarrow p$ od**
  this temporally links nodes that are the same in terms of the type, owner and attribute slots.

These constructor templates are used in the examples described in this chapter. The now-connections cannot be reflexive. now-link and now-ref describe the single parent case just uses the type and owner fields. The now-relationship describes the two parent version also uses the attribute field to place an extra constraint on connection formation. The node type $R$ stands for result node which has a two valued belief value for {watch, ignore}.

The construction rules express the essential details but do not describe how the connections are made, for this we first need a description of the DDN graph structure. The DDN is composed of temporally separated buckets (or subgraphs) that are interconnected, with nodes at the graph vertices accessed via their respective bucket. Each bucket has a set of vertex-indexes $V$ and edges that are denoted by the set of edge-destinations $ED$ and the set of edge-sources $ES$. $ED$ and $ES$ are subsets of the set of graph indexes $GI$, such that each graph index is defined so that in addition to referring to the vertex-index they
also refer to the relative time, e.g., if \( g \in GI, h \in H, v \in V \) then \( g = (h, v) \) where \( h \) is a deictic-time-index relative to \( t_{\text{now}} \).

Denoting edges with edge-destinations and edge-sources enables us to describe the links between buckets. In the examples in this chapter we will use only three buckets, however, a general definition is:

**Definition 5.2** A DDN graph \( \text{DDNG} = (TS, \{s_{g_1}, \ldots, s_{g_n}\}) \) where \( TS \) is a list of bucket-indexes ordered to reflect their current temporal status. Each bucket \( s_{g_i} = (v_i, e_{s_i}, e_{d_i}) \) where \( v_i \subseteq V \) are \( s_{g_i} \)'s vertices, \( e_{s_i} \subseteq GI \) are \( s_{g_i} \)'s edge-sources, and \( e_{d_i} \subseteq GI \) are \( s_{g_i} \)'s edge-destinations.

This definition of directed edges decomposes the meaning of the \((\to)\) function into two distinct functions which need to be executed to add an edge. These functions are \text{add-edge-destination}(\text{DDNG, NODE, TO-NODE})\) and \text{add-edge-source}(\text{DDNG, NODE, FROM-NODE})\), they both operate on graph-indexes and are of type \( \text{DDNG} \to GI \to GI \to \text{DDNG} \).

### 5.2.3 Graph properties

The complexity of the graph is related to the length of temporal history reasoned over. A graph that has no temporal history just reasons about the current time value. Increasing the length of history, also increases the size of the graph. If we assume that the number of nodes, \( n \), at time \( t_{i-1} \) is the same as at time \( t_i \) then we have a linear increase in the number of nodes as the length of history, \( h \), in increased, i.e., \( O(n \times h) \). Neapolitan [172, page 249] (also see Pearl [183, page 174]) describes that in the case of an arbitrary singly connected network, the time requirement for the update of all variables is linear with respect to the number of variables in the network. This means that the DDN scales well with respect to the length of temporal history.

All the MACDDN construction macros, except \text{now-relationship}, generate \( O(n) \) algorithms. \text{now-relationship} generates an \( O(n \log n) \) algorithm, this is because we have a Cartesian product of \( p \) and \( q \), and can ignore the symmetric component. This means that the generation of all the links between the nodes in the DDN graph is \( \Theta(n \log n) \).

The number of nodes in the DDN at time \( t \), is determined by the results from the preattentive operators, which are themselves dependent on the number of scene objects.

We can ensure that no loops are formed in the DDN graph at runtime if (1) we only allow non-temporal connections to be made between nodes that share the current time value, and (2) that no loop is formed by the directed edges added by the one-step-temporal-connections. The first part is present in the use of now-connections. The second is a static test that can be performed prior to runtime. These constraints restrict the DDN graphs to tree-like forms.

### 5.2.4 Updating algorithm

Updating the \text{DDNG} takes three stages as described in the NEI algorithm given in figure 5.6.
The first stage involves running the connection rules and is called update-graph-structure. It consists of two components defined in the MACDDN language called node-building and node-linking that are used to describe which node tuples are to be created and how they are to be connected.

The second stage involves constructing an equivalent Bayesian network, and then setting it up by identifying the appropriate conditional probability matrix for each processor node in the Bayesian network. The belief slots from the DDNG node tuples provide the initial values for the root nodes in the Bayesian network.

In the third stage the inference algorithm is run, with the results from this used to update the belief values of selected result nodes in the DDNG.

The following algorithm description expands the NEI algorithm from figure 5.6 and is represented pictorially in figure 5.8.

```
procedure NEI
  input DDNG
  (1) run update-graph-structure on DDNG
  (1.1) run node-building
  (1.2) run node-linking
        name the updated graph DDNG'
  (2) create equivalent Bayesian network "BayesNet" for DDNG'
  (2.1) construct BayesNet from DDNG'
  (2.2) setup BayesNet by adding conditional probability
        matrices and initialising root and evidence nodes
  (3.1) run inference-algorithm on BayesNet
  (3.2) update value slots in DDNG' to give DDNG"
  output DDNG"
```

We provide examples of the first two stages in section 6 when we describe appropriate node-building and node-linking functions and the conditional probability matrices for the mutual-proximity test.

The DDN provides part of the preattentive mechanism in HIVIS-WATCHER. The belief values of the result nodes held in the DDN provide likelihood measures of which buffer indexes (from a result node generated by now-ref) or pairs of indexes (from a result node generated by now-relationship) are worth further attention. This further attention is achieved by the allocation of an agent, a limited resource, that we want applied to the objects that are the most relevant to the current official-observer task.

5.2.5 Discussion

The graph structure formed over successive update operations is determined by the nodes that are present in the buckets and the connection rules. The graph topology may contain subgraphs that are repeated on successive graph updates because the connection rules remain constant. Although, the DDN does support topologically changing graph structures, in an implementation it would be advantageous to make use of the local structural continuity however, this also causes additional complexity and has not been investigated.
In this version of the NEI algorithm we separate the DDN into two parts: the graph and the Bayesian network. These are really one and the same. We have separated them because the execution form of the Bayesian network contains additional structure that is not present in the DDN graph, and in this initial implementation modelling of the graph structure was a necessary first step. As shown in figure 5.8 separating the two allows the inference mechanism to be implemented as a stand alone piece of code. Integrating the two would provide a more efficient implementation, because it could take advantage of local structural continuity, and is a subject for future work.

The implementation of the DDN described here does not use clique trees (for details see Tarjan and Yannakakis [226] and Pearl [183]). Dean et al. [54] used clique trees in their DDN as a useful approach for reducing graph complexity that works well with graphs that have a fixed, temporally-repeated structure. However, we have described a more dynamic, graph restructuring approach that we found was required to model the set of object interactions. In the more general multiple-connected-networks, an approach like clique trees would be needed to enable Bayesian inference to be viable, although generating the cliques a new for each graph update provides an additional computational overhead.

5.3 Tasknet

The use of a task-based Bayesian network is mentioned in Rimey and Brown [200] where it is used to actively direct the camera using geometric relationships. Here, however, our application requires scene surveillance based on spatial (and temporal) reasoning relative to a static camera, and we are guiding the computation performed according to the selected surveillance task.

One of the surveillance tasks requires a solution to the spatial arrangements problem. The DDN is used to identify relevant pairs of objects that are produced in the graph by the MACDDN template now-relationship. This relationship is also used to identify
which agency results should be composed. The composition resolves the deictic viewpoints of the selected pair of agents to provide the official-observer interpretation of the relationship. This composition uses the mechanism described in chapter 4 section 7. The resulting composite is input to the pair's dedicated TASKNET which builds a coherent interpretation of the temporally evolving object relationship.

The scene objects are represented in the DDN by their buffer slot indexes, which provide the owner symbols for the node tuples. Each identified pair of buffer slot indexes \((i, j)\), is given a distinct TASKNET which we can denote TASKNET\(_{(i, j)}\). The TASKNETs have a temporally fixed tree structure with conditional probabilities that are defined before runtime to reflect a preferential bias towards a feature that is deemed to be most interesting. The input nodes represent key features relevant to the task the TASKNET has been constructed to identify. The output root node represents the overall belief, based on the evidence collected so far, in a set of candidates consisting of the wanted task, related but unwanted tasks, and the default unknown task.

Allocating an attentional process reflects the three stages in the allocate component.

- The focus-of-attention stage is described by the focus function in table 5.4 which first ensures that on allocation the TASKNET is initialised and collecting information related to the pair, and then ensures that an agent is running on both selected objects.

- The selective-attention stage is described by the continue function which updates the root nodes and runs the inference algorithm.

- When an uninteresting situation is recognised, we first instantiate the belief value held in the pair's current DDN reference node via function M2 in figure 5.5. On the next clock tick when the DDN is updated and the inference algorithm run, if the relationship is still present, a high ignore value is placed in the reference node at what is now \(t_{\text{now}}-1\). This will ensure that, unless a potentially interesting indicator is present at time \(t_{\text{now}}\), the relationship will be ignored. This high ignore value is temporally propagated that causes the allocated agents to be terminated, and when this occurs we also terminate the TASKNET for the pair which is described by the terminate function in table 5.4.

Using this method, once a relationship is identified it can be ignored or continue to be watched.

We can only attend to a limited number of objects and use a utility function to determine whether a particular object is worth attending. The utility involves the value of its worth (the interestingness) and the cost of performing the attention operators.

\[
U(\alpha) = \frac{V(\alpha)}{C(\alpha)}
\]

To simplify the current implementation we assume that the cost is constant, so we can just use

\[
U(\alpha) = V(\alpha) = \arg\max_{\gamma \in \text{INTERESTING-INDEXES}} P_{\gamma}(\text{watch})
\]
function focus (i,j)
   (1) let TN be a TASKNET (take from the freelist or create)
   (2) initialise the TN
   (3) update root nodes with the results from combining those of the agents on i and j
   (4) run inference-algorithm on TN
   (5) return belief value held by the likely-episode result node in TN
   (6) save TN at store address (i,j)

function continue (i,j)
   (1) let TN be the TASKNET at store address (i,j)
   (2) update root nodes with the results from combining those of the agents on i and j
   (3) run inference-algorithm on TN
   (4) return belief value held by the likely-episode result node in TN
   (5) save TN at store address (i,j)

function terminate (i,j)
   (1) put TASKNET at store address (i,j) on freelist
   (2) set store address (i,j) to Ø

Table 5.4: Algorithms focus, continue and terminate for TASKNET.

Note that in this case ignore and watch are the two states of a single binary variable denoting interestingness, i.e.,

\[ P_\gamma(\text{ignore}) = 1 - P_\gamma(\text{watch}). \]

In the following examples this variable is denoted by R, since it is typically a result node.

5.4 MECHANISM

The two stage algorithm is given in figure 5.9 which expands upon the top level loop given in figure 5.1. The two stage DDN and TASKNET attentional mechanism each take a system-cycle to perform and are temporally overlapped to take advantage of the current visual-operator outputs. In the DDN and TASKNET approach changing the connection rules run by the DDN is likely to have a major effect on system behaviour, where as, changing the TASKNET graph is likely to have less effect. The TASKNET is more for fine tuning the behaviour of the attentional mechanism, for differentiating between similar surveillance tasks. The overall goal of the project is the formation of conceptual descriptions that capture what is happening in the scene. We would like this conceptual description to evolve in conjunction with the scene events and not be dependent upon the observation of a whole episode, we are making a distinction between saying what happened and what is likely to be happening now.

5.5 SUMMARY

This has covered details of our approach to implementing ALLOCATE and COLLECT. There are likely to be other valid approaches, but we selected the Bayesian approach mainly because it addresses the uncertainty present in the data we are using, and provided a
<table>
<thead>
<tr>
<th>STAGE ONE</th>
<th>STAGE TWO</th>
</tr>
</thead>
<tbody>
<tr>
<td>• (1) Update WORLD with the next image frames pose positions.</td>
<td></td>
</tr>
<tr>
<td>• (2) Run selected visual operators.</td>
<td></td>
</tr>
<tr>
<td>• (3.1) Allocate agents.</td>
<td>(iv) Supply attentional visual-operator results to appropriate agent.</td>
</tr>
<tr>
<td>(i) Peripheral visual-operator outputs provide marker related primitive features as input to DBN.</td>
<td></td>
</tr>
<tr>
<td>(ii) Call node-building, then node-linking, extract tree structured Bayes net and run inference alg. to update DBN nodes.</td>
<td></td>
</tr>
<tr>
<td>(iii) Select most interesting marker-pair relationship called ((i,j)).</td>
<td></td>
</tr>
<tr>
<td>• (3.2) Run agent on each selected marker to (i) select appropriate attentional visual-operators and (ii) determine deictic events.</td>
<td></td>
</tr>
<tr>
<td>• (3.3) Collect results.</td>
<td>(v) If both agents in marker-pair ((i,j)) provide events, update their TASKNET.</td>
</tr>
<tr>
<td></td>
<td>(vi) Run inference algorithm on TASKNET.</td>
</tr>
<tr>
<td></td>
<td>(vii) Use results from TASKNET’s likely-episode node (LE_{(i,j)}) to update the DBN relationship node (R_{(i,j)}).</td>
</tr>
<tr>
<td>• (4) Run the effector system.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.9: The two stage algorithm.

Graph based representation we could extend to model dynamic changes in scene data. The high-level vision system described here has been implemented in Lisp. This implementation has been used in the following experiments.

5.5.1 Experiments

To illustrate how the `GLOBAL-FORM` and `LOCAL-FORM` operate we use three examples called: proximity, coordination and grouping. The initial example illustrates how DDN, TASKNET and AGENCY operate. It also highlights some limitations with the approach taken, requiring an extension to be made to the `GLOBAL-FORM` which we call the “KERNEL”. These extensions are demonstrated in the coordination and grouping examples that illustrate multiple-object interactions and the reintroduction of more globally oriented reasoning.
CHAPTER 5. ATTENTIONAL CONTROL

6 PROXIMITY

The examples described in this section use the mutual-proximity algorithm outlined in section 5. Here we add further details and use it to identify when the system perceives the passage of two proximate objects travelling in the same direction. Once identified as being present, the pairing can be used for further reasoning.

We are using two separate applications of Bayesian networks. First we describe how the DDN is used to represent the current proximity relationships. The second is used to assess the task relatedness of the actions selected so far, providing a task-directed attentional mechanism.

We have chosen functions for OP1 and OP2 that will be used in the example given in section 6.3. These functions are used for spatial reasoning about the visual data. For our simple operation, we recall that in chapter 4 section 6 the primitive property proximity was given, together with a mapping of qualitative values to integers. These values [0, ..., 4] are the range used by $N_t(t)$ in table 5.5 (implementation details are given in appendix C section 3.2). For the complex operation, recall (from chapter 4 section 6) the event detection that uses this proximity measure to determine the bearing of the nearest (other) object relative to the object's frame-of-reference. The result is a deictic reference such as behind-me, beside-me or infront-of-me. (Note that conditions (1, 2, 3, 4) of definition 5.1 are fulfilled — the complex operator only works when given the result saying which, if any, is the nearest object.)

6.1 SELECTING INTERESTING FEATURES

To combine the proximity information that develops over time we have used a dynamic form of decision network (DDN) similar to Bayesian networks. That is, the network evolves structurally over time to capture the changing proximity relationships between scene objects. In the case outlined here, the graph formed describes which object has a proximate neighbour. The node values hold data that contributes towards a measure reflecting how interesting this relationship is to the official-observer watching the scene. The basic graph primitive is depicted as:

\[ N_A(t) \xrightarrow{R_{(A,B)}(t)} N_B(t) \]

This graph describes a mutual-proximity relationship between objects $A$ and $B$, capturing the primitive notion of object $A$ being near $B$ and $B$ being near $A$. The relationship node $R$ denotes this mutual-proximity and holds a belief value that reflects how interesting this is. The states and purpose of the two node types are given in table 5.5. The directed edges hold the fixed conditional probabilities given in table 5.6. These matrices were derived from careful consideration of the possible input values. The spatial relationship is specified by the matrix $M_{RN}$, capturing the belief that a proximity relationship becomes increasingly more interesting as two objects are identified as being nearer to each other. Figure 5.11(a) shows that when we evolve the network over time we make use of maintained
Table 5.5: The DDN node types.

\[
M_{\text{RN}} = \begin{array}{c|cc}
\text{ignore} & \text{watch} \\
\hline
\text{not-near} & 1.0 & 0.0 \\
\text{nearby} & 0.4 & 0.6 \\
\text{close} & 0.3 & 0.7 \\
\text{very-close} & 0.2 & 0.8 \\
\text{touching} & 0.0 & 1.0 \\
\end{array}
\]

\[
M_{\text{RR}} = \begin{array}{c|cc}
\text{ignore} & \text{watch} \\
\hline
\text{ignore} & 0.9 & 0.1 \\
\text{watch} & 0.1 & 0.9 \\
\end{array}
\]

Table 5.6: DDN conditional probabilities for the proximity mechanism. Note that (1) \(M_{\text{RN}}\) is not used directly, and is present to define \(M_{\text{RNN}}\); (2) details of the CJP function are given in appendix D; and (3) on the right are illustrations of the subgraphs for which child node conditional probability matrices are provided.

official-observer object references to the image data. If, at the current time point, \(A\) and \(B\) are still near to each other we can form temporal links from the previous node to the current ones as shown in figure 5.11(b). From the relative temporal position of the previous time point the temporal links are to the next time point. These directed temporal edges each hold one of the matrices representing temporal continuity, here this is either \(M_{\text{NN}}\) for proximity, or \(M_{\text{RR}}\) for the pairwise relationship. Figure 5.11(c) shows how we extract a singly connected network (SCN) from the DDN graph which is done for the parsimonious reason that a tree is easier to evaluate than a multiply connected network. This graph simplification preserves the results obtained from the previous value propagation allowing them to contribute effectively to the formation of the new belief about the relationship that now holds between object \(A\) and \(B\). Figure 5.11(c) is a simplification of the more complete network structure shown in figure 5.10. In the example developed in this paper each SCN produced only holds values from the current and previous time points. If the DDN graph contains more than one relationship, then a SCN is formed for each relationship. Once all propagations have completed the most interesting relationship can be selected.
Figure 5.10: $N_A(t-1)$ and $N_B(t-1)$ hold values of nearness from time $t-1$, $R_{(A,B)}(t-1)$ is the result from the previous run of update, $N_A(t)$ and $N_B(t)$ are the new values. Run the inference engine to update the beliefs of the chance nodes and finally obtain a value for $R_{(A,B)}(t)$.

Figure 5.11: Graph linking structure.

Figure 5.12: Change over time.
The DDN network changes structurally over time to reflect the relationships between the objects of interest. For example, figure 5.12 shows what happens if \( N_B \) is missing for two time points \( t3 \) and \( t4 \) due, say, to occlusion and the nearest neighbour to \( N_A \) now becomes \( N_C \). When \( N_B \) is detected again it is identified as being nearest and the new links and nodes in the network are created to reflect this.

### 6.1.1 Implementation Details

To test for mutual-proximity we use the two parent rule `now-relationship`, where the attribute field holds the name of the most proximate neighbour. To express this we will use the node types \( N \) and \( R \) for `nearest-object-to-me` and `mutual-proximity-relationship`, with the variables \( x \) and \( y \) used to denote any two distinct objects (i.e., \( x \neq y \)), then the necessary MACDDN rules are:

<table>
<thead>
<tr>
<th>constructor macro</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>now-connections</td>
<td>[(\forall x, y \in ABS1, now-relationship(\text{(N^x_{(x,y)})}, \text{(N^y_{(x,y)}})) \rightarrow add\text{-}vertex(\text{(R^x_{(x,y)})}, \text{(R^y_{(x,y)})}, \text{(N^x_{(x,y)})} \rightarrow \text{(N^y_{(x,y)})}) \wedge \text{(N^x_{(x,y)})} \rightarrow \text{(R^x_{(x,y)})}, \text{(N^y_{(x,y)})} \rightarrow \text{(R^y_{(x,y)})})]</td>
</tr>
<tr>
<td>one-step-temporal-connections</td>
<td>[(\forall x \in ABS1, temporal-continuity(\text{(N^x_{(x)})}) \rightarrow R^x_{(x)}), (\forall s \in ABS2, temporal-continuity(\text{(R^s_{(s)})}) \rightarrow R^s_{(s)}]</td>
</tr>
</tbody>
</table>

Where ABS stands for the currently active buffer slots, with \( \text{ABS1} \subseteq \text{OV1} \) and \( \text{ABS2} \) being the powerset of \( \text{ABS1} \) such that \( \text{ABS2} \subseteq \text{OV2} \).

As described in section 5.2.4, we use two components called node-building and node-linking. Here we will define the algorithms that result from expanding the MACDDN constructor macros. These algorithms use a different language where node-type \( \in \{ "nearest", "mutual-proximity" \}\), such that "nearest" corresponds to \( N \), and "mutual-proximity" corresponds to \( R \). Also these algorithms use a slightly different notation for denoting delictic temporal references, with `now` corresponding to \( t_{\text{now}} \), and `prev` corresponding to \( t_{\text{now}-1} \). The node-linking algorithms are given in figure 5.13, and the node-building algorithm is:

```plaintext
procedure node-building(DDN, WORLD, ASPECTS)
    let SG be the current-subgraph of DDN
    for each INDEX in active-buffer-slots of WORLD do
        let OTHER be peripheral-aspect(ASPECTS, INDEX, "nearest")
        if OTHER \( \neq \) nil do
            let PAIR be list(INDEX, OTHER)
            let NODE be make-node("nearest", PAIR)
            add-vertex(SG, NODE)
        od
    od
    return DDN
```

\(^3\)In the Lisp implementation the node-types are keyword symbols.
In node-building the data-structure access function active-buffer-slots(WORLD) returns a list of active object indexes and the function peripheral-aspect(ASPECTS, INDEX-REFERENCE, OPERATOR-TYPE) returns the result created for INDEX-REFERENCE by visual-operator OPERATOR-TYPE and stored in ASPECTS.

Once node-building is done, we do node-linking to update the edges in DDN. As shown in figure 5.13, this updating is performed by a collection of procedures defined by MACDDN constructor macros and called by node-linking. The looping over mutual-proximity-relationship can be made more efficient by taking account of its symmetrical nature. Once node-linking has finished, the result is a graph primitive like that shown in figure 5.11(c).

The DDN itself does not hold an overall picture of what is happening. For the creation of a fuller picture we use a separate Bayesian network called TASKNET.

6.2 Meta Level

In section 5.3 we described how a TASKNET can be allocated to a pair of indexes to provide a description of how observed properties relate to defined behaviour. In this section we use the TASKNET to distinguish between likely overtaking and following behaviour. In addition to the deictic-positional relationships (described in chapter 4 section 6.1.1) we also use speed difference (see section 6.1.3) and heading difference (see section 6.1.2). The speed and heading values shown in figure 5.14 are obtained when an agent is allocated to an object that has a most proximate neighbour. This is illustrated in figure 5.14 by the summary on the right. At the bottom of this pictorial summary the DDN graph provides preattentive selection of the two deictic agents. The results from the mutual-proximity nodes in the DDN (index pairs of the form (i, j)) are used to select which agents to run on which indexes.

In figure 5.14, next up from this, the events generated by the typical-object-model run on each agent, and the composition of these agent results produces a composite feature value, which is integrated together with other evidence to produce an estimate of the likely episode that the two agents are engaged in. For example, the positional feature obtained from the agent results has two components: the index-reference of the nearest object; and the deictic-event-primitive ∈ {behind-me, beside-me, infront-of-me, blocking-me}.

The input given to TASKNET(\(i,j\)) uses the composition via the respective composition matrix \(\oplus\) shown at the top left of figure 5.14 (such as that formed from the composition rules given in table 4.9 of chapter 4 section 7). We can combine the results from the correct agents running on indexes \(i\) and \(j\), because \(i\) and \(j\) are in the scope of the GLOBAL-FORM. In this example the TASKNETs have a very simple structure consisting of four nodes, one of each type given in table 5.7. \(EV_{(i,j)}\) is a root node, \(LE_{(i,j)}\) is the output node, and they are related according to the conditional probabilities given in table 5.8. The result of running the Bayesian inference algorithm is used to guide agent allocation as described earlier by identifying the likelihood that the relationship is either overtaking, following, queuing or unknown.
<table>
<thead>
<tr>
<th>constructor macros</th>
<th>algorithm generated</th>
</tr>
</thead>
</table>
| now-relationship($x^{(n)}_{(x^{(y)})}$) | procedure mutual-proximity-relationship($SG_{now}$, A, B)  
if (node-type(A) = "nearest") and  
(node-type (B) = "nearest") and  
(node-owner (A) = reverse(node-owner(B))) do  
let R be make-node("mutual-proximity",  
node-owner(A))  
let RGI be make-graph-index(now, R)  
let AGI be make-graph-index(now, A)  
let BGI be make-graph-index(now, B)  
add-vertex($SG_{now}$, R)  
add-edge-source($SG_{now}$, RGI, AGI)  
add-edge-destination($SG_{now}$, AGI, RGI)  
add-edge-source($SG_{now}$, RGI, BGI)  
add-edge-destination($SG_{now}$, BGI, RGI)  
od |
| temporal-continuity($x^{(n)}_{(c)}$) | procedure near-temp-continuity($SG_{prev}$, $SG_{now}$, A, B)  
if (node-type(A) = "nearest") and  
(node-type(B) = "nearest") and  
(node-owner(A) = node-owner(B)) do  
let AGI be make-graph-index(prev, A)  
let BGI be make-graph-index(now, B)  
add-edge-source($SG_{prev}$, AGI, BGI)  
add-edge-destination($SG_{now}$, AGI, BGI)  
od |
| temporal-continuity($x^{(n)}_{(c)}$) | procedure mutual-prox-temp-cont($SG_{prev}$, $SG_{now}$, A, B)  
if (node-type(A) = "mutual-proximity") and  
(node-type(B) = "mutual-proximity") and  
(node-owner(A) = node-owner(B)) do  
let AGI be make-graph-index(prev, A)  
let BGI be make-graph-index(now, B)  
add-edge-source($SG_{prev}$, AGI, BGI)  
add-edge-destination($SG_{now}$, AGI, BGI)  
od |
| | procedure node-linking(DBN)  
let $SG_{now}$ be current-subgraph(DBN)  
let $SG_{prev}$ be previous-subgraph(DBN)  
let $V_{now}$ be $SG$-vertices($SG_{now}$)  
let $V_{prev}$ be $SG$-vertices($SG_{prev}$)  
for each B in $V_{now}$ do  
for each A in $V_{now}$ do  
mutual-proximity-relationship($SG_{now}$, A, B) od  
for each A in $V_{prev}$ do  
near-temp-continuity($SG_{prev}$, $SG_{now}$, A, B)  
mutual-prox-temp-cont($SG_{prev}$, $SG_{now}$, A, B)  
od od |

Figure 5.13: The node-linking rules. Here mutual-proximity-relationship is a now-connection rule and both near-temp-continuity and mutual-prox-temp-cont are one-step-temporal-connection rules. Also, in the definition of mutual-proximity-relationship the reverse function is used to switch around the elements in the (INDEX, OTHER-INDEX) list created by node-building.
6.3 Examples

Here we will look at road traffic domain examples of tasks like "look for likely overtaking behaviour" or "look for likely following behaviour". These examples are to show how task bias can affect the results observed from the same sequence of data.

In the data set occlusion scenario (shown in appendix A), so named because it that contains a sequence where a car is occluded by a lorry. The features of observer control that should be present here is the ability to maintain an agent on an interesting object (e.g., the lorry and overtaking car).

6.3.1 Overtaking

The purpose of this example is to show that the DDN and TASKNETs can pick out a pair of vehicles that are performing an overtaking episode. To do this we will use the TASKNET policy "attend to likely overtaking and ignore likely following". This "overtaking" is likely to be non-intentional behaviour which just depends upon how the vehicles on the road are organised (and contributes to the uncertainty of identification).
Table 5.7: The TASKNET node types.

<table>
<thead>
<tr>
<th>NODE</th>
<th>STATES AND FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>( EV_{(i,j)} )</td>
<td>(back-to-back, trans-overtaking-back, following, back-blockage, trans-overtaking-front, possible-blockage, overtaking-passing, head-blockage, head-on, impasse) The composite event of the marker pair ((i,j)).</td>
</tr>
<tr>
<td>( LE_{(i,j)} )</td>
<td>(overtaking, following, queuing, unknown) The likely-episode of the marker pair ((i,j)).</td>
</tr>
<tr>
<td>( RHD_{(i,j)} )</td>
<td>(parallel, right-angles, heading) The relative heading difference of the marker pair ((i,j)).</td>
</tr>
<tr>
<td>( SPD_{(i,j)} )</td>
<td>(both-stopped, same, difference) The relative speed difference of the marker pair ((i,j)).</td>
</tr>
</tbody>
</table>

\[
M_{LEBV} =
\begin{pmatrix}
\text{overtk} & \text{follow} & \text{queue} & \text{unk} \\
\text{back-to-back} & 0.0 & 0.0 & 0.0 & 1.0 \\
\text{trans-overtaking-back} & 1.0 & 0.0 & 0.0 & 0.0 \\
\text{following} & 0.0 & 1.0 & 0.0 & 0.0 \\
\text{back-blockage} & 0.0 & 0.2 & 0.3 & 0.5 \\
\text{trans-overtaking-front} & 1.0 & 0.0 & 0.0 & 0.0 \\
\text{possible-blockage} & 0.0 & 0.2 & 0.2 & 0.6 \\
\text{overtaking-passing} & 1.0 & 0.0 & 0.0 & 0.0 \\
\text{head-blockage} & 0.0 & 0.2 & 0.3 & 0.5 \\
\text{head-on} & 0.0 & 0.0 & 0.0 & 1.0 \\
\text{impasse} & 0.0 & 0.1 & 0.5 & 0.4
\end{pmatrix}
\]

\[
M_{LEBSPD} =
\begin{pmatrix}
\text{both-stopped} & \text{follow} & \text{queue} & \text{unk} \\
\text{same} & 0.0 & 0.5 & 0.5 \\
\text{different} & 0.5 & 0.3 & 0.1 & 0.0
\end{pmatrix}
\]

\[
M_{LBRHD} =
\begin{pmatrix}
\text{overtk} & \text{follow} & \text{queue} & \text{unk} \\
\text{parallel} & 1/3 & 1/3 & 1/3 & 0.0 \\
\text{right-angles} & 0.0 & 0.0 & 2.2 & 0.8 \\
\text{head-on} & 0.0 & 0.0 & 0.0 & 1.0
\end{pmatrix}
\]

\[
M_{EVR} = \forall o, i \in EV, P(o | i) = \begin{cases} 0.73 & \text{if } o = i \\ 0.03 & \text{if } o \neq i \end{cases}
\]

\[
M_{SFDPD} = \forall o, i \in SPD, P(o | i) = \begin{cases} 0.8 & \text{if } o = i \\ 0.1 & \text{if } o \neq i \end{cases}
\]

\[
M_{RHD} = \forall o, i \in RHD, P(o | i) = \begin{cases} 0.8 & \text{if } o = i \\ 0.1 & \text{if } o \neq i \end{cases}
\]

\[
M_{LEBV RHD SPD} = \operatorname{CIF}(M_{LEBSPD}, M_{LEBHD}, M_{LEBV})
\]

Table 5.8: TASKNET conditional probabilities for the road traffic example.

The results from the frame sequence in figure 5.15 are both incorporated in the figure and also shown in table 5.9. The vehicle shapes given in outline denote uninteresting peripheral objects, the number near each vehicle is it's index reference (or buffer slot number), and the filled in shapes are selected objects that have been allocated to an agent. The sequence begins with two vehicles being selected because they are near each other. At frame 24, they are identified as following one another and are ignored. In frame 48, the pair (3,2) are selected because they were near each other in frame 36. In frames 48 to 72 the pair (3,2) are not mutually close together. At frame 84, possible overtaking or following is identified and identifying overtaking is given preference since it is the task
<table>
<thead>
<tr>
<th>time</th>
<th>pairs</th>
<th>watch</th>
<th>agents</th>
<th>referb events</th>
<th>likelihood values</th>
<th>likely episode</th>
<th>ignore</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(1.0)</td>
<td>0.699</td>
<td>0 1 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>(1.0)</td>
<td>0.719</td>
<td>0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>(1.0)</td>
<td>0.779</td>
<td>0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>(1.0)</td>
<td>0.807</td>
<td>0 1</td>
<td>2 1 behind-me</td>
<td>0.164 0.003 0.088 0.655</td>
<td>unknown</td>
<td></td>
</tr>
<tr>
<td>(3.2)</td>
<td>0.599</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>(1.0)</td>
<td>0.820</td>
<td>3 0 1</td>
<td>1 2 infront-of-me 2 1 behind-me</td>
<td>0.098 0.818 0.035 0.049</td>
<td>following t</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>(1.0)</td>
<td>0.303</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>(1.0)</td>
<td>0.542</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>(3.2)</td>
<td>0.599</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>(1.0)</td>
<td>0.599</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3.2)</td>
<td>0.909</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>(3.2)</td>
<td>0.977</td>
<td>1 2 3</td>
<td>1 2 beside-me 2 1 beside-me 2 1 beside-me</td>
<td>0.632 0.041 0.039 0.288</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>(3.2)</td>
<td>0.987</td>
<td>2 3 3</td>
<td>1 2 beside-me 2 1 beside-me 2 1 beside-me</td>
<td>0.860 0.056 0.035 0.049</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>(3.2)</td>
<td>0.981</td>
<td>2 3 3</td>
<td>1 2 beside-me 2 1 beside-me 2 1 beside-me</td>
<td>0.860 0.056 0.035 0.049</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>168</td>
<td>(3.2)</td>
<td>0.977</td>
<td>2 3 3</td>
<td>1 2 beside-me 2 1 beside-me 2 1 behind-me</td>
<td>0.860 0.056 0.035 0.049</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>(3.2)</td>
<td>0.977</td>
<td>2 3 3</td>
<td>1 2 infront-of-me 2 1 beside-me 2 1 behind-me</td>
<td>0.479 0.437 0.035 0.049</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>(3.2)</td>
<td>0.977</td>
<td>2 3 3</td>
<td>1 2 infront-of-me 2 1 beside-me 2 1 behind-me</td>
<td>0.479 0.437 0.035 0.049</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>204</td>
<td>(3.2)</td>
<td>0.977</td>
<td>2 3 3</td>
<td>1 2 infront-of-me 2 1 beside-me 2 1 behind-me</td>
<td>0.479 0.437 0.035 0.049</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>216</td>
<td>(3.2)</td>
<td>0.967</td>
<td>2 3 3</td>
<td>1 2 infront-of-me 2 1 beside-me 2 1 behind-me</td>
<td>0.479 0.437 0.035 0.049</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>228</td>
<td>(3.2)</td>
<td>0.961</td>
<td>2 3 3</td>
<td>1 2 infront-of-me 2 1 beside-me 2 1 behind-me</td>
<td>0.479 0.437 0.035 0.049</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>240</td>
<td>(3.2)</td>
<td>0.961</td>
<td>2 3 3</td>
<td>1 2 infront-of-me 2 1 beside-me 2 1 behind-me</td>
<td>0.479 0.437 0.035 0.049</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>252</td>
<td>(3.2)</td>
<td>0.961</td>
<td>2 3 3</td>
<td>1 2 infront-of-me 2 1 beside-me 2 1 behind-me</td>
<td>0.479 0.437 0.035 0.049</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>264</td>
<td>(3.1)</td>
<td>0.676</td>
<td>3 4 3</td>
<td>2 0 infront-of-me</td>
<td>0.437 0.035 0.049</td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>276</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>288</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>(3.1)</td>
<td>0.599</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4.2)</td>
<td>0.609</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>312</td>
<td>(3.1)</td>
<td>0.719</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4.2)</td>
<td>0.719</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.9: The results from looking for likely overtaking behaviour. The agent names are given as numbers and displayed in figure 5.15 as shaded in outlines: 0 is dark-gray, 1 in medium grey, 2 is light-grey. The numbers in the agent columns are buffer slot indexes, and are also displayed in the figure. The "referb events" use the-referb's field values and list the marker names for self and other, together with the event that relates self and other. The □ indicates no currently available agent reference.
Figure 5.15: Selecting which pairs of vehicles are involved in overtaking.

objective. During frames 96 and 108 one of the vehicles occludes the other from the camera\textsuperscript{4}. By frame 132 overtaking is positively identified.

In table 5.9 the missing entries are because of the occlusion where no mutually proximate objects are visible to the observer. The numbers given are for the indexes, not the agents. The same preattentive operator, with the result from TASKNET causing different forms of attentional behaviour e.g., terminate and continue. In figure 5.16 we provide a comparison between overtaking and following.

\textsuperscript{4}Figure 3.10 shows the camera’s field-of-view and, although less obvious, the camera position does affect the contents of the frame updates because we are dependent upon what is visible from the camera position not what is visible from the overhead view.
### Table 5.10: The results from looking for likely following behaviour.

<table>
<thead>
<tr>
<th>time</th>
<th>pairs</th>
<th>watch</th>
<th>agents</th>
<th>refobj events</th>
<th>likelihood values</th>
<th>likely episode</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>overt</td>
<td>follow</td>
</tr>
<tr>
<td>0</td>
<td>(1.0)</td>
<td>0.599</td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>(1.0)</td>
<td>0.719</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>(1.0)</td>
<td>0.779</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>36</td>
<td>(1.0)</td>
<td>0.807</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>48</td>
<td>(1.0)</td>
<td>0.820</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>60</td>
<td>(1.0)</td>
<td>0.868</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>72</td>
<td>(1.0)</td>
<td>0.888</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>84</td>
<td>(3.2)</td>
<td>0.599</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>108</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>(1.0)</td>
<td>0.599</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>132</td>
<td>(3.2)</td>
<td>0.977</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>144</td>
<td>(3.2)</td>
<td>0.783</td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>156</td>
<td>(3.2)</td>
<td>0.960</td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>168</td>
<td>(3.2)</td>
<td>0.676</td>
<td></td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>180</td>
<td>(3.2)</td>
<td>0.938</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>192</td>
<td>(3.2)</td>
<td>0.676</td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>204</td>
<td>(3.2)</td>
<td>0.838</td>
<td></td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>216</td>
<td>(3.2)</td>
<td>0.685</td>
<td></td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>228</td>
<td>(3.2)</td>
<td>0.871</td>
<td></td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>240</td>
<td>(3.2)</td>
<td>0.644</td>
<td></td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>252</td>
<td>(3.2)</td>
<td>0.859</td>
<td></td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>264</td>
<td>(3.1)</td>
<td>0.676</td>
<td></td>
<td>3</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>276</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>288</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>(3.1)</td>
<td>0.699</td>
<td></td>
<td>3</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>312</td>
<td>(3.1)</td>
<td>0.719</td>
<td></td>
<td>3</td>
<td>1</td>
<td>a</td>
</tr>
</tbody>
</table>

#### 6.3.2 Comparison between overtaking and following

In figure 5.16 to save space the tracks have been stacked into short conduit like segments of spacetime. The three pictures for overtaking repeat (and extend) the sequence shown in figure 5.15. Alongside these three pictures are three pictures that illustrate the difference in agent allocation when we change the TASKNET policy to “attend likely following and ignore likely overtaking”. Comparing table 5.9 with table 5.10 shows the effect of TASKNET on the interpretation of the same data. This example illustrates how a given surveillance task can affect interpretation (see chapter 2 section 4.2.2) and also that uncertainty and ambiguity of interpretation is present even after scene objects have been identified. These examples illustrate how task based control enables HLVIS-WATCHER to obtain a viable task dependent description of what is happening in the scene.
Figure 5.16: Comparison between overtaking and following.
6.3.3 LIMITATIONS

These examples illustrate how the implemented computational theory operates on the occlusion scenario. However, there is a hidden problem in that the implementation cannot distinguish between the overtaking and overtaken objects. This is a problem inherent in the use of the agent viewpoint as seen by the perceiver, because the integration of viewpoints loses position specific details in the formation of the mutual proximate relationship. This limitation is illustrated in the two worst case situations given in figure 5.17. The data gathered by the local-deictic results is not enough to describe what is happening in the scene, although is does provide a solution to the spatial arrangements problem.

The limitations described here are due to the use of local spation-temporal reasoning. The typical-object-model does not retain a temporal history of past events, and because of this we are not able to identify accomplishments (this depends on being able to recognise that the situation after the state change is different from the situation before the state change). An illustration of this temporal history limitation problem is given in figures 5.17 and 5.18. Consider figure 5.18 where we have the situation of one car overtaking another, if we have A behind B, then A beside B, then A behind B again we would like to say that overtaking has not occurred because there has been no accomplishment of passing.

6.4 SUMMARY

The results from the proximity example show how HIVIS-WATCHER is able to alter the attention of agents to fulfill a particular task, and say when the task’s behaviour is taking place. The difference between the task of “look for following” and “look for overtaking” are represented by changes made to the behaviour of TASKNET. Overtaking and following are similar behaviour, they are both based upon the same preattentive primitive cue of mutual-proximity and because of this one can easily be proposed as an instance of the other. This similarity of behaviour requires the ability to terminate-attention of the undesired behaviour. Terminate-attention causes the current attended to objects to be ignored, however, if they are the only interesting objects in the field-of-view the system returns to look at them. If there were other equally interesting activity taking place then terminate-
<table>
<thead>
<tr>
<th>cue</th>
<th>attend</th>
<th>ignore</th>
<th>see table</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutual proximity</td>
<td>overtaking</td>
<td>following</td>
<td>5.9</td>
</tr>
<tr>
<td>mutual proximity</td>
<td>following</td>
<td>overtaking</td>
<td>5.10</td>
</tr>
<tr>
<td>gross change in motion</td>
<td>giveaway</td>
<td></td>
<td>5.14</td>
</tr>
<tr>
<td>gross change in motion</td>
<td></td>
<td></td>
<td>5.15</td>
</tr>
<tr>
<td>mutual proximity</td>
<td>overtaking</td>
<td>following</td>
<td>5.17, 5.18</td>
</tr>
</tbody>
</table>

Table 5.11: The policies.

attention would cause one (or more) of these to become the focus-of-attention via the reallocation of agents.

We developed the overtaking and following example on the occlusion sequence and have also tested it on different sequences such as the one illustrated in figure 5.32 and table 5.17 which, as shown in table 5.11, are results from running HIVIS-WATCHER with the same policy.

The elements of the approach described here to express the task based control model that we described in section 4 of ALLOCATE and COLLECT include: the use of agent markers and the typical-object-model to express routines of how object behaviour is interpreted; the development of the DDN for temporal guidance; and the separation of preattentive and attentive operators. These all contribute towards the situated approach developed here, endowing HIVIS-WATCHER with timely response to the surveillance tasks described here. The demonstrated results more closely correspond to the “here-and-now” quality of surveillance, than the accumulation of events and episodes provided by HIVIS-MONITOR.
7 Kernal

In chapter 2 section 3.2.3 on the deictic viewpoint we introduced the difference between local and global representation and reasoning with reference to figure 2.9. Both forms of representation and reasoning are held by the perceiver, and in the previous sections of this chapter we have investigated the use of the local-form for the interpretation of scene object behaviour. We have found the problems with this approach are two fold: although useful for describing single or spatially related binary objects, it is not suitable for coordinating non-local relationships, and the expression of object behaviour as an identifiable episode may take longer than can be easily modeled using the local-form of representation. The first problem is another version of viewpoint integration and the second is the temporal history limitation identified in section 6.3.3. To address these we need to use more global representation and reasoning that can (1) operate over longer episode-length time spans, (2) a spatial representation to coordinate perceiver references to scene object and other parts of the environment, (3) the ability to reason about these perceiver references. To provide these we have added the "Kernal" which extends the representation of the perceived environment in the virtual-world, the operators in the peripheral-system and the global-form in the central-system.

7.1 Perceived Environment

To coordinate perceiver references in the scene we need the global frame-of-reference provided by the global-form. We do not require great spatial resolution as the objective here is not representation of the physical shape of scene objects and ground-plane regions. This more detailed description is provide by the local-form. In the global-form we only require a coarse representation of position that can be used to access the local-form information should this be required. This global coordinate system uses \( \mathbb{Z}^2 \) providing a "grid" over the ground-plane that has a qualitative description of object position, an illustration of this is given in figure 4.46 of chapter 4 section 5. The global viewpoint of the scene conforms to the intuitive idea of when describing the scene in global terms we use general and abstract properties, where as in local terms we would use more detailed object oriented properties.

7.2 Markers and Operators

In section 6 we use one type of marker called "agent", here we introduce three new types of marker, each of which runs a different set of rules. This contrasts with the agent markers which all ran the same set of rules that we called the typical-object-model. Introducing these new types of marker has required changing how attentional markers are displayed, and as shown in table 5.12, we have adopted Agre and Chapman's [3, 33] use of simple geometric shapes. The position of a marker is determined by its address in the grid, and for display this is mapped (scaled, etc) to provide an indication of its place on the ground-plane or scene object to which it has been allocated. In section 2.1 we described how agent
Table 5.12: Attentional marker types. This replaces the shading of objects in section 6. In effect they are both the same and the use of shapes is just for conformity.

Markers can only be attached to buffer slots (called tracking). The new markers, called "kmarkers", have additional functionality in that they can also be placed at a grid address. Each kmarker has with it an associated set of uses, that reflect how it is incorporated in the kernel routines. Not all kmarkers are used all of the time, their selection is task specific. Kmarkers are used by the operators described in chapter 4 section 6.4.2, that are able to move a kmarker, cause a kmarker to track an object, access properties about the environment (grid position) being marked, etc.

7.3 Global Form

To reason about the placement of kmarkers we use the kernel rules in the central-system. This reasoning is expressed as routines that are used to determine if a particular form of "policy" behaviour has taken place. Each stage in the routine corresponds to an accomplishment of observed object behaviour, and the transition through each stage of the routine in the correct sequence describes an episode. The cellwise-time language (see appendix C section 1.1) provides the representation of accomplishment we use here. This is similar to the notion described by Dretske [62, page 35] where a "process" is seen as producing or causing a terminal state called the "product". As shown in figure 5.19 which illustrates the meaning of an accomplishment in the cellwise-time language and also shows the correspondence to Dretske's terminology: a Dretske process is an activity and a Dretske product is the state (or possibly beginning of the next activity) after a state-change. Some state-changes are contextual important and we call these events.

The addition of the kernel rules does not affect the homogeneity of the central-system (see chapter 2 section 4.3.5). These rules add more detail to the model of the perceiver, describing its situatedness, that reflect the observer's behaviour (an important component of the visual process, see chapter 2 section 4.2.2). For pragmatic reasons the kernel is described as a separate module however, it does not partition memory. We have just described another facet of what is available to the perceiver.
7.4 IMPLEMENTATION DETAILS

The KERNEL itself has a PERIPHERAL- and CENTRAL-SYSTEM split, and these components are integrated into those of HIVIS-WATCHER to form part of the top level loop shown in figure 5.1. The execution of the KERNEL's CENTRAL-SYSTEM circuit network runs in parallel with each of the agent executions and has its effector requests combined with those from the agents. There is no operator conflict because KERNEL and AGENCY use different sets of operators.

The operators in the KERNEL's PERIPHERAL-SYSTEM use a much simplified representation of the data held in the virtual-world than that used by the AGENCY. This simplified representation is more like the video game representations used by Agre and Chapman. An example is given in figure 5.20, showing the ground-plane from Bremer Stern. In the figure each grid element is a 2m × 2m square of the ground-plane and the six attentional kmarkers are shown in their "home" position (the bottom-left of the grid). Kmarkers can only be placed at grid positions and be made to track another marker (e.g., an agent) by the KERNEL. To perform tracking of an agent marker we require an interface to how agents are allocated. We described in section 4.2 how agents are allocated via ALLOCATE, and data concerning the allocation of agents is input to the KERNEL rules via a set of kernel-interface-wires that can be set from agent-operators. (An alternative (and perhaps better) approach would be to use the marker-m-assigned? operators, described in chapter 4 section 6.4.2, but this would require an extra clock-tick to obtain the necessary properties.)

7.5 SUMMARY

In this section we have described the extension to HIVIS-WATCHER that we found required to provide a more detailed global viewpoint. We demonstrate the use of this extended GLOBAL-FORM with examples in the following two sections.
8 Coordination

To illustrate the need for local and global viewpoints we describe how the policy "look for likely giveaway behaviour" can be performed. This provides an example that raises new issues that were not relevant to the policies for likely overtaking and following. We begin by describing a new preattentive cue that is used by allocate and then outline the kernel rules that extend the central-system.

8.1 Gross change in motion

The preattentive cue we are going to use is called "gross-change-in-motion". It is based on the spatio-temporal discontinuity operator we described in chapter 4 section 6.2.2, and is used to enable allocate to assign an agent marker to any scene object that changes from moving to stationary, or stationary to moving. This cue attracts us to instances where there is a gross change in motion, such as, when a vehicle stops at a junction to assess the road conditions, or when people get up after sitting in the same place for some time.

To denote the current typical motion of an object we use a prior called motion-prior such that motion-prior ∈ \{moving, stationary\}. When the prior value is moving, then stopping is unusual; when the prior value is stationary, moving is unusual. We assume that normal behaviour of an object for motion-prior is moving. If the observed motion of an object is different from the motion-prior then abnormal behaviour is taking place (see chapter 2 sections 3.2.4 and 4.2). If this abnormal behaviour persists it becomes the norm and the motion-prior is changed to reflect this.

To define this as a preattentive cue we need to specify a set of MACDDN construction rules for the DDN. These rules use three node types PS, PM, and R, which stand for motion-prior value is stationary, motion-prior value is moving, and result. The R node is present to hold the belief value for \{watch, ignore\} that represents the result of running the inference-algorithm. The MACDDN rules are as follow:

<table>
<thead>
<tr>
<th>constructor macro</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>now-connections</td>
<td>( PS_{(x)} \rightarrow R_{(x)} )</td>
</tr>
<tr>
<td>( \forall x \in \text{ABS1}, \text{now-ref}(PS_{(x)}) )</td>
<td>( PM_{(x)} \rightarrow R_{(x)} )</td>
</tr>
<tr>
<td>one-step-temporal-connections</td>
<td></td>
</tr>
<tr>
<td>( \forall x \in \text{ABS1}, \text{temporal-continuity}(R_{(x)}) )</td>
<td>( R_{(x)} \rightarrow R_{(x)} )</td>
</tr>
<tr>
<td>( \forall x \in \text{ABS1}, \text{temporal-continuity}(PS_{(x)}) )</td>
<td>( R_{(x)} \rightarrow PS_{(x)} )</td>
</tr>
<tr>
<td>( \forall x \in \text{ABS1}, \text{temporal-change}(PS_{(x)}, PM_{(x)}) )</td>
<td>( PS_{(x)} \rightarrow PM_{(x)} )</td>
</tr>
<tr>
<td>( \forall x \in \text{ABS1}, \text{temporal-change}(PM_{(x)}, PS_{(x)}) )</td>
<td>( PM_{(x)} \rightarrow PS_{(x)} )</td>
</tr>
<tr>
<td>( \forall x \in \text{ABS1}, \text{temporal-continuity}(PM_{(x)}) )</td>
<td>( PM_{(x)} \rightarrow PM_{(x)} )</td>
</tr>
</tbody>
</table>

As before ABS stands for the currently active buffer slots, with \( \text{ABS1} \subseteq \text{DV1} \). This set of rules is independent of those for mutual-proximity. In this example we do not need to determine mutual-proximity, and which means that the rules in the DDN for this example
cannot detect overtaking or following. We could use both sets of rules together as they do not conflict, allowing us to look for both gross-change-in-motion and overtaking should we wish to do so. The addition of different preattentive connection rules in the DDN changes the behaviour of the system.

The now-connections are for combining the current and previous values for the interestingness of this object and the node type reflects the motion-prior of the object. When an object is moving a PM node is used and when an object is stationary a PS node is used. To illustrate how these rules operate consider the sequence of graphs shown in figure 5.21 where an object changes from moving to stationary to moving again. Along the top of the figure are the frame time-cells, with the two buckets now holding the contents of the $t_{\text{now}}$ time-cell (i.e., the time of the current frame) and $\text{prev}$ holding the contents of the $t_{\text{now-1}}$ time-cell. In addition to the two buckets for now and prev, the figure also shows the nodes that are in these buckets and the links between and inside them. Notice how the contents of the $t_1 \text{ now}$ bucket become the $t_{i-1}'s \text{ prev}$ bucket, and that the links are changed to conform to the connection rules. Table 5.13 describes the conditional probability matrices and joint probability matrices, and in the “Node type allocation” table we describe the patterns of graph node to which these various matrices are assigned.

The results shown in figure 5.22 and table 5.14 use the same sequence as in the proximity example, as shown by tables 5.9 and 5.10. Notice the difference in agent allocation which is due to the use of a different cue, and also the lack of TASKNET results because the allocated agent does not produce any events that need to be temporally integrated. This example was just to demonstrate the gross-change-in-motion cue and how it can be used to allocate agents to objects. Comparing table 5.14 with tables 5.9 and 5.10 shows that the difference in attentional performance by the official-observer is much more pronounced than changing attend and/or ignore in the policy, which we illustrated in the proximity example. Although we are not modelling scan paths, these different amounts of change in attentional performance has a correspondence to Yarbus’ [260] experimental results described in chapter 2 section 4.2.2. The difference in attentional performance between the proximity example and gross-change-in-motion is because the surveillance tasks are based on different preattentive primitives, one is on grouping and the other on spatio-temporal discontinuity. This difference in preattentive cues may account for the

<table>
<thead>
<tr>
<th></th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>gross motion</td>
<td>moving</td>
<td>moving</td>
<td>stationary</td>
<td>stationary</td>
<td>moving</td>
</tr>
<tr>
<td>now bucket</td>
<td>$\text{R} \rightarrow \text{PM}$</td>
<td>$\text{R} \rightarrow \text{PM}$</td>
<td>$\text{R} \rightarrow \text{PS}$</td>
<td>$\text{R} \rightarrow \text{PS}$</td>
<td>$\text{R} \rightarrow \text{PM}$</td>
</tr>
<tr>
<td>prev bucket</td>
<td>$\text{R} \rightarrow \text{PM}$</td>
<td>$\text{R} \rightarrow \text{PS}$</td>
<td>$\text{R} \rightarrow \text{PS}$</td>
<td>$\text{R} \rightarrow \text{PS}$</td>
<td>$\text{R} \rightarrow \text{PS}$</td>
</tr>
</tbody>
</table>

Figure 5.21: A sequence of graph examples to illustrate how gross change in motion affect the allocation of PM and PS nodes.
Table 5.13: DDN conditional probabilities for gross change in motion. Note that MO denotes the general description of motion and is used by the subgraphs described in the "Node type allocation" table.

difficulty people have in interpreting the occlusion scenario, when after looking at the overtaking episode (see appendix A) to also identify the relevance of the waiting vehicle. This may be because this part of the scenario, that is insignificant under one surveillance task, becomes unexpectedly significant under another surveillance task, rather like how changing the expectation of the perceiver viewing the duck-rabbit figure (see chapter 2 figure 2.12 and also the text example in figure 2.13) affects its interpretation. Next we will use the preattentive cue developed here to identify vehicles that are waiting at a junction for a gap in the traffic.

8.2 GIVEWAY

We have now defined the preattentive cue for identifying a vehicle that could potentially be giving way to some other scene object. We will use this to investigate what GLOBAL-FORM functionality is necessary to perform the policy of “look for giveaway behaviour”. An important difference between this coordination example and the proximity one, is that the identification of a giveaway episode involves a number of distinct script-like steps, involving different scene participants. To achieve the identification of a giveaway episode we need to coordinate the perceptual actions performed by the official-observer. In the case of the giveaway-episode we have the following script-like stages: the initial cue is a stationary, or slowing down object; the duration of the object being stationary in a giveaway-zone makes giveaway more likely; the object is giving way to something; and the object moves when the obstruction is gone.
Figure 5.22: The results from looking for gross change in motion.
Table 5.14: The results from looking for gross change in motion.

When the official observer perceives an object that appears to give way, it is assuming that the intention of the object is to continue progress once the way is clear. The detection of this episode involves attending to stationary vehicles that may be giving-way to something, and then finding out what that something may be. When the two roles of the giving-way and giving-way-to have been found, an area of mutual conflict can be identified (the space in front of the stationary vehicle and through which the giving-way-to moving objects will pass). This area links the stationary object to its cause. All that remains is to determine that the stationary vehicle is giving way to approaching traffic, and exhibits no other plausible behaviour (e.g., broken down, parked). In this description of the giveaway episode there are three important entities: the first two correspond to the two roles in the giveaway episode and are denoted by $S$ for the-stationary-vehicle, and $PB$ for the-vehicle-that-$S$-is-giving-way-too; and the third is denoted CA for the-conflict-area (a special region).

8.2.1 The Perceiver Routines

We separate the giveaway episode into five routines that use region-based-prediction (see appendix C section 2.2.4), and perceiver level coordination. These routines are:

- Notice-stopping-object, which on completion generates event-gw1. The gross change in motion from moving to stationary allocates an agent, called $S$, and prompts the question
“why is vehicle $S$ stationary?” There are a number of possible answers however, if $S$ is in a give way zone of an entry lane to a roundabout, the most likely answer is that $S$ is giving way to something on the roundabout.

- Look-for-path-blocker, which on completion generates $\text{event-gw}2$. This stage identifies $PB$. For $PB$ to be blocking $S$ it does not need to be physically in the way, it can also block by having “right-of-way” such that its path will block $S$. If $PB$ exists it will typically be in the give way-zone corresponding to the give way-zone that $S$ is occupying. This is illustrated in figure 5.23 and described in chapter 3. Figure 5.24 illustrates vehicles in the give way region (we return to the multiple-object case in section 8.2.3). Using contextual-indexing we find $PB$. Having proposed that $PB$ is blocking $S$’s path causing $S$ to avoid a collision by stopping, the next two routines are to prove that this is true.

- Work-out-conflict-area, which on completion generates $\text{event-gw}3$. Having predicted the paths of $S$ and $PB$, we intersect them to find the mutually shared conflict area, $CA$. The presence of $CA$ supports the proposal made in stage path-blocked, and binds both $S$ and $PB$ together. $CA$ should be invariant during the give way episode.

- Watch-for-enter-conflict-area, which on completion generates $\text{event-gw}4$. In order to determine whether $S$ gives way to $PB$, we wait until $PB$ has passes through $CA$. To determine that $S$ is giving way, we only need to check that at least one object passes through $CA$.

- Notice-starts-to-move, which on completion generates $\text{enter-gw}5$. We then observe if $S$ moves. The gross change in motion from stationary to moving reallocates an agent to $S$.

The five routines can describe a temporal sequence of perceiver activity and although we concentrate on the order that identifies a give way episode, routines are intended to be more generally applicable and, for this reason, to fully identify a give way episode we also need to check that the routines were performed in one continuous sequence. The temporal order in which the routines occur is important. Watch-for-enter-conflict-area should occur
before notice-starts-to-move, else a giveaway cannot be said to be taking place, and in this situation, depending upon their mutual proximity a collision may result.

In routine work-out-conflict-area, as the vehicle positions are updated, the vehicles involved do not create changing areas of path intersection because the boundary of the coarse-grained prediction is defined by the road surface. The \( \mathcal{C}A \) is shared by \( \mathcal{S} \) and \( \mathcal{PB} \), with \( \mathcal{PB} \) making steady progress towards \( \mathcal{C}A \), and \( \mathcal{S} \) remaining stationary. \( \mathcal{C}A \) is represented by an activation-plane.

The detection of the giveaway episode makes use of a number of extensions that use global storage available to operators and described in chapter 4 section 6.3.

The operators in the kernel's peripheral-system include:

\[
\text{(marker-in-giveaway-region? marker doit?) accesses SPATIAL-LAYOUT to determine if marker is on an object that is in a giveaway region and if this giveaway region has a giveaway-to-region. Sets the wires *registered-marker-in-giveaway-region*, *marker-in-giveaway-region*, *registered-marker-has-giveaway-to-region* *marker-has-giveaway-to-region*. This operator also generates kernel event event-gw1.}
\]

\[
\text{(region-occupied? region marker doit?) allocates the given marker to the object nearest the front of the region (according to the typical flow of objects through the region). Sets the wires *registered-region-occupied?*, and *region-occupied?-object*. This operator generates event-gw2.}
\]

\[
\text{(marker-path-intersect? marker1 marker2 region1 region2 result doit?) creates the path predictions for marker1 and marker2 using activation regions region1 and region2, intersects region1 and region2 putting the result in activation region result. Sets the wires *registered-marker-path-intersect?*, *registered-conflict-area*. Generates event-gw3.}
\]

These operators are specific to the giveaway episode and are added to the more general ones described in chapter 4 section 6.4.2.

The kernel rules for giveaway in the kernel's central-system include the following. The routine is initiated by gross-change-in-motion and an agency typical-object-model operator publicize-stationary-object! that sets the wires *registered-the-stationary-object* and *the-stationary-object* that hold a boolean and an agent name \( m \ (m \in \{0, 1, 2\}) \).

\[
\text{(arbiter warp-marker! (move-marker to-marker doit?)}
\]

\[
\text{(propose-default noop :move-marker (constant #f))}
\]

\[
\text{(propose stationary :move-marker *stationary-marker* :to-marker *the-stationary-object* :doit? (constant #t))}
\]

\[
\text{(condition stationary (andg *registered-the-stationary-object* *run-giveaway-mechanism* (invert *stationary-marker-assigned*))))}
\]

Once the *stationary-marker* is assigned and if *marker-in-giveaway-region*
has not been registered we find out the region type property of the object that *stationary-marker* is marking.

(propose-default noop :marker (constant *f*) :doit? (constant *f*))
(propose in-grp :marker *stationary-marker* :doit? (constant *t*))
(condition in-grp (andg *stationary-marker-assigned* (invert *registered-marker-in-giveaway-region*))))

If *stationary-marker* is in a giveaway region we run the region-occupied? arbiter to determine if the giveaway-to-region is occupied.

(propose-default noop :region (constant *f*) :doit? (constant *f*))
(propose find-grp :region *marker-has-giveaway-to-region* :marker *head-marker* :doit? (constant *t*))
(condition find-grp (andg *registered-marker-has-giveaway-to-region* *run-giveaway-mechanism*)))

Once we have *stationary-marker* and *head-marker* assigned to objects we use three activation planes (see chapter 4 section 6.4.2) to hold the path predictions, and the result from their intersection.

(propose-default noop :marker1 (constant *f*) :marker2 (constant *f*) :region1 (constant *f*) :region2 (constant *f*) :result (constant *f*) :doit? (constant *f*))
(propose find-CA :marker1 *stationary-marker* :marker2 *head-marker* :region1 *activation-plane1* :region2 *activation-plane2* :result *conflict-plane* :doit? (constant *t*))
(condition find-CA (andg *stationary-marker-assigned* *head-marker-assigned* (invert *registered-conflict-area*))))

We repeatedly test to find out when *head-marker* enters the conflict-area. This test is terminated once the object *stationary-marker* is on moves. We detect this by *stationary-marker* becoming unassigned, because *stationary-marker* was attached by warp and not track it does not follow the movements of the object that it currently points at.

(propose-default noop :marker (constant *f*) :region (constant *f*) :doit? (constant *f*))
(propose dotest :marker *head-marker* :region *conflict-plane* :doit? (constant *t*))
(condition dotest (andg *registered-conflict-area* *head-marker-assigned*)))

These arbiters and operators are demonstrated in the following example.

### 8.2.2 Example

The results from the giveaway implementation are given in table 5.15 and figure 5.25. The table shows the HIVIS-WATCHER only uses three attentional markers to perform the giveaway detection routine. Also the events listed in this table correspond to the five routines described in section 8.2.1. The table also corresponds to table 5.14 which contains the results for looking for gross-change-in-motion, notice that the column for *agent2* is the same.
Figure 5.25: The results from looking for the giveaway episode. Note that *head-marker* is the triangle (marker 4) and *stationary-marker* is the cross (marker 5). A key to the attentional markers:

<table>
<thead>
<tr>
<th>markers</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>shape</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>agents</td>
<td>kmarker</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A table describing the regions used by the three activation planes in this example:

<table>
<thead>
<tr>
<th>frame 192</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>global variable</th>
<th><em>activation-plane1</em></th>
<th><em>activation-plane2</em></th>
<th><em>conflict-plane</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>the-stationary-vehicle's--path-prediction</td>
<td>the-blocking-path-prediction</td>
<td>the-conflict-area</td>
</tr>
<tr>
<td>display</td>
<td>black</td>
<td>dark grey</td>
<td>light grey</td>
</tr>
</tbody>
</table>
Table 5.15: The kernel results from looking for the giveaway episode.

<table>
<thead>
<tr>
<th>time</th>
<th>agents</th>
<th>kernel</th>
<th>events</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>168</td>
<td>1</td>
<td>1</td>
<td>(EVENT-GW1)</td>
</tr>
<tr>
<td>180</td>
<td>3</td>
<td>1</td>
<td>(EVENT-GW2)</td>
</tr>
<tr>
<td>192</td>
<td>3</td>
<td>1</td>
<td>(EVENT-GW3)</td>
</tr>
<tr>
<td>204</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>216</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>228</td>
<td>1</td>
<td>1</td>
<td>(EVENT-GW4)</td>
</tr>
<tr>
<td>240</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>252</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>264</td>
<td>1</td>
<td>1</td>
<td>(EVENT-GW5)</td>
</tr>
<tr>
<td>276</td>
<td>1</td>
<td>1</td>
<td>(EVENT-GW5)</td>
</tr>
<tr>
<td>288</td>
<td>1</td>
<td>1</td>
<td>(EVENT-GW6)</td>
</tr>
<tr>
<td>300</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>312</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.16: A model of increasing the likelihood of a giveaway episode as being observed as the events are observed in sequence.

The frames 108, 132–156 describe the allocation of *agent2* cued by gross-change-in-motion. At frame 120 the vehicle moved again, before the motion-prior was altered from moving to stationary by the agency operator change-motion-prior! The value of motion-prior is changed by frame 168 because the object ceases to have an interesting motion property. Frame 192 shows the results from the region path predictions that generate the contents of the kernel activation planes. Frames 204–258 display the activation plane for CA. Frame 228 shows the removal of *head-marker* following a successful intersection with CA. Frame 264 shows the object pulling away from underneath *stationary-marker* and in frame 276 *agent2* is again allocated to the object because of its gross-change-in-motion.

8.2.3 Refinements

The perceiver routine outlines the approach taken in the example described here. There are a number of refinements that could be performed in a future implementation to make this routine more robust. For example:

---

5Note that the selection of *agent2* is not essential to the operation of this routine. It is only selected again because the agency free-list has been implemented as a stack.
- There may be more than one stationary object, such as in the two cases illustrated in figure 5.24. In the first case, the current implementation could be improved so that $S$ represented the group of stationary vehicles. The situation in the second case may not be so relevant, since the giveaway episode of the car behind will occur after the giveaway episode of the car in front.

- As in the example used here there may be more than one blocking vehicle, indeed there maybe a stream of vehicles (similar to the problem illustrated in figure 5.17). In the current implementation, to simplify the problem of representing $PB$, we just pick the nearest object to $S$ that is in the giveaway-to-zone. It would be better to select the group and reason about the group, an approach that we discuss in section 9. The case where a stream of vehicles continually blocks $S$ complicates the group model, because to model this requires the representation of a deictic object (the group) that has changing elements.

- To derive the likelihood of a giveaway episode we need to extend the implementation described here so that it can reason about the sequential ordering and completeness of the event sequence. To do this we could employ one of the event-composition approaches described in chapter 4 section 3 such as the use of weights or finite-state-machines. An example of this is given in table 5.16 which illustrates how the likelihood of the detection of a giveaway episode could be modelled.

### 8.3 Summary

This coordination example shows that HIVIS-WATCHER can identify likely giveaway behaviour. We have demonstrated how the gross-change-in-motion preattentive operator can be used as the foundation for identifying the first and last routines in the temporal history of a giveaway episode. These two events are used to associate the episode with the stationary vehicle and, as shown in figure 5.26, they also act as boundaries to the the giveaway episode. The three central routines of the giveaway episode demonstrate how the kernel is able to coordinate information about different scene objects. This is in contrast to the local reasoning of the typical-object-model. This example has shown how the
LOCAL-FORM and GLOBAL-FORM can be combined to provide a task dependent interpretation of the occlusion scenario. Using the task of "look for likely giveaway behaviour" we have obtained a different valid interpretation of the same data that was used in the proximity example. Although not demonstrated here, it is possible for HIVIS-WATCHER to "look for both giveaway and overtaking (and/or following) behaviour". This is because the preattentive operator used for both the proximity example and the coordination example use different sets of MACDDN construction rules that do not conflict.

Before developing the solution described here, the initial motivation was to find out if the typical-object-model could be extended to accommodate more complex reasoning about other scene objects. However, although the perceiver has access to each agent, the individual agents do not share or exchange information, which made it difficult to combine the evolving data about the objects identified as performing the giveaway episode. The development of the gross-change-in-motion preattentive operator and KERNEL provides a more elegant solution with the perceiver routine and typical-object-model having separately defined purposes: that of modelling how the perceiver coordinates its perceptual skills, and knowledge about the behaviour of participants in the world.
Chapter 5. Attentional Control

Figure 5.27: The object that provided the front edge is first and the object that provided the back edge is last.

Figure 5.28: Cluster life.

Figure 5.29: Cluster example.

9 Grouping

This third example addresses the temporal history limitation problem identified in section 6.3.3. To do this we develop a computational model for the preattentive operator, called “clustering”, which we introduced in chapter 4 section 6.2.3 as a gestalt primitive that identifies groups of objects. After this we present two examples that use grouping. The first example is a more complete solution to the surveillance task of identifying overtaking and that is able to distinguish between overtaking and similar behaviour such as drawing-abreast-and-pull-away (which is illustrated in figure 5.18). The second example demonstrates some initial work concerning queues.

9.1 Clusters

We do not fully develop an algorithm or model for clustering here, this is only an initial framework developed to explore its use. Figure 5.27 shows a cluster as being a convex hull that encloses a group of proximate objects. The two object case is similar to the mutual proximity relationship, and we make use of this similarity in section 9.2 A cluster is a markable entity, adding a level of hierarchy to the marker methodology similar to the FINST model described by Pylyshyn [190]. In addition to the proximity criterion, an additional constraint is that all objects in a cluster should also share the same relative heading, providing mutual spatial and temporal continuity, this is called the “cluster-requirement”. There are four phases in the life of a cluster, called “apart”, “join”, “together”, and “leave”. The relationships between these phases is illustrated in figure 5.28, with “apart” being the prior value. Figure 5.29 illustrates some of the following possibilities.

- One or more objects join the cluster such that they fulfill the cluster-requirement of this cluster. This is an example of the join phase.
- One or more objects leave the cluster, but the cluster retains a quorum of members. An example of a combination of the leave and join phases.
- A cluster is created with a join phase.
- The cluster dissolves. None of the members conform to the cluster-requirement, producing possibly multiple leave phases.
- The cluster remains intact, this is the together phase, providing the cluster-requirement is be maintained.

This initial model does not fully capture the hierarchical quality of object relationships within a group and the different levels of conformity to the cluster-requirement. This model also seems to complex for a preattentive operator. A better model is described by Watt [246], but this has not been used in the current implementation.

Once we have obtained the convex hull outline of the group boundary, and have the heading of the group from its cluster-requirement, we can use these to place an ordering over the contents of the group. This ordering enables us to identify the front and back elements of a cluster. Figure 5.27 shows how the objects face properties are used to identify these extreme positions within the group because the object that provided the front edge is first and the object that provided the back edge is last. We use this test for position within a group in the first grouping example.

9.2 OVERTAKING

In this grouping example we illustrate how the temporal history limitation problem can be solved. To do this we use the accomplishment framework described in section 7.3 where we introduced Dretske’s term “product”, for denoting the terminal state following an accomplishment. In addition to the temporal history limitation problem, we also address the loss of position specific details caused by the positional compositional matrix described in section 6.2. To describe the approach taken here, we will use an example based on figure 5.18 and which is illustrated in figure 5.30. This approach requires at least three time-cells, that in figure 5.30 are called \( t_n \), \( t_n + \delta \), and \( t_n + \delta + \Delta \). These three time-cells enable us to describe an accomplishment with \( (t_n) \ldots (t_n + \delta) \) being the activity and \( (t_n + \delta + \Delta) \) being the Dretske product. In figure 5.30, case (a) indicates the Dretske
product for the completion of overtaking. This is the case that we want to distinguish
from case (b) where no change in the order of the two vehicles has occurred.

Figure 5.31 shows how we detect this change in ordering by using two kmarkers called
*tail-marker* (▼) and *head-marker* (▲). We use three routines:

- Notice-the-overtaking-pair, where TASKNET identifies two objects that are likely to
  be overtaking.
- Identify-head-and-tail, where we allocate the two kmarkers to describe the positional
  ordering given at time \( t_n \).
- Watch-for-switch-in-order, where we monitor the ordering of the kmarkers until some
  change takes place, such as that given at \( t_n + \delta + \Delta \) in state (c). State (d) does not
  describe a change and is not recognised as anything significant because nothing has been
  accomplished.

The ordering of the markers is not dependent upon their relative or absolute positions
in the perceivers frame of reference. Their ordering is only dependent upon the position
of the objects within the group, and the intrinsic frame-of-reference of the objects in the
group. During overtaking both objects are heading in the same direction, with spatio-
temporal continuity, fulfilling the cluster-requirement.

9.2.1 Implementation details

The implementation of the kernel extensions described here builds upon the implementation
of proximity given in section 6. Here we will use the policy (cue = mutual proximity,
attend = overtaking, ignore = following) that represents "looking for likely overtaking
behaviour" (see table 5.11 for a comparison of policies). In addition to the kernel operators
described in section 4 section 6.4.2 we also need to be able to construct a group from
a set of agent markers. The agents in this example are selected by their mutual proximity
relationship node in the DDN and so we are just making explicit the preattentive
group information represented by the mutual proximity relationship. Given a pair of agent
markers, we create a group by accessing the properties of the objects that they point at
and cluster them to identify the front and back elements as shown in figure 5.27.

(make-group elements doit?) forms a group from the list of elements, and
sets the wires *group-front-marker*, *group-front-marker-assigned*,
*group-back-marker*, *group-back-marker-assigned*.

(event-switch-order doit?) generates the kernel event event-SO1.

This completes the kernel's peripheral-system. To complement this the kernel's
central-system is extended to include the following arbiters. The wires *overtaking-
pair-assigned* and *overtaking-pair* are set once TASKNET has identified a pair
of objects that are engaged in likely overtaking behaviour.
To assign *head-marker* and *tail-marker* on the clock-tick we use two track! operators and arbiters.

Once *head-marker* and *tail-marker* are assigned we need to check when *tail-marker* is coincident with the agent on the object at the front of the group, and when *head-marker* is coincident with the agent on the object at the back of the group. This requires two coincident? operators and arbiters so that we can perform both tests on each clock-tick.

The two coincident tests do the work of identifying when, and if, the two agents involved in likely overtaking do actually overtake. To generate an event relating such a happening we use.

When the overtaking pair of agents are no longer assigned we finish this routine.
(arbitr end-group (marker1 marker2 doit?)
(propose-default noop :marker1 (constant *f*) :marker2 (constant *f*)
  :doit? (constant *f*))
(propose finish-group :marker1 *head-marker* :marker2 *tail-marker*
  :doit? (constant *t*))
(condition finish-group (andg *full-overtaking-check*
  (invert *overtaking-pair-assigned*))))

This version of HIVIS-WATCHER incorporating these KERNEL operators and arbiters is demonstrated in the following example.

9.2.2 Example

To illustrate how this approach determines whether overtaking occurs we will use a sequence of frames from the enter scenario. This sequence is not interrupted by an occlusion and, incidentally, was also used in chapter 4 to provide figures 4.44 and 4.43. Figure 5.32 and table 5.17 describe results similar to the proximity example given in section 6, except here the agents are indicated by a diamond and a pentagon (for *agent1* and *agent2*). Table 5.17 details their allocation, the events and the identification of likely episodes. When overtaking is identified a group is formed from the two objects that the agents point to, and once the "head" and "tail" agents are identified the *head-marker* and *tail-marker* are allocated. As shown in table 5.18 when *head-marker* is found not to be marking the head agent of the group the event "switch-order" is generated. The production of this event denotes the accomplishment of overtaking between the two objects in the group.

In figure 5.32 frames 260–290 show the initial selection of proximate objects. At frame 290 a different pairing is found to be better, with the results generated in table 5.17 reflecting this change in allocation because *agent2* is no longer assigned to an object. At frame 300 the TASKNET identifies two objects engaged in likely overtaking behaviour and a frame 320 has informed the KERNEL of its findings. The kernel forms a group and allocates the two KERNEL markers to identify the front and back elements of the group. At frame 360 a switch in ordering is identified, and at frame 370 one of the vehicles goes out of the camera's field-of-view. This frame sequence demonstrates the usefulness of extracting information about global spatial properties (i.e., clusters) that are implicitly associated with the scene objects. The results also show that the test using kmmarkers is more sensitive than the agent results, and that the KERNEL can model temporal continuity via kmmarker allocation. This grouping example also provided another demonstration of coordination between the AGENCY and KERNEL.

The results in the tables show that the AGENCY identifies an overtaking episode and that the KERNEL identifies the switch in order. These are two separate processes that have a one-way relationship. The operation of the AGENCY is independent of the KERNEL, while the KERNEL is dependent on the AGENCY for more detailed analysis of object behaviour in the scene.
Figure 5.32: Checking for overtaking.

Note that *tail-marker* is ▼, *head-marker* is △, and the two agent markers are *agent1* ◆, and *agent2* ■.
Table 5.17: The TASKNET results from looking for likely overtaking. Note that the □ indicates no currently available agent reference.

<table>
<thead>
<tr>
<th>time</th>
<th>pairs</th>
<th>watch</th>
<th>agents</th>
<th>reobj events</th>
<th>refobj event</th>
<th>likelihood values</th>
<th>likely episode</th>
<th>ignore</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>(1.0)</td>
<td>0.845</td>
<td><img src="image" alt="agent symbols" /></td>
<td><img src="image" alt="event symbols" /></td>
<td></td>
<td>overtaking</td>
<td></td>
<td></td>
</tr>
<tr>
<td>260</td>
<td>(1.0)</td>
<td>0.934</td>
<td><img src="image" alt="agent symbols" /></td>
<td>0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>270</td>
<td>(1.0)</td>
<td>0.960</td>
<td><img src="image" alt="agent symbols" /></td>
<td>0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>280</td>
<td>(1.0)</td>
<td>0.963</td>
<td><img src="image" alt="agent symbols" /></td>
<td><img src="image" alt="event symbols" /></td>
<td>0 1</td>
<td><img src="image" alt="likelihood values" /></td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>290</td>
<td>(2.0)</td>
<td>0.845</td>
<td><img src="image" alt="agent symbols" /></td>
<td><img src="image" alt="event symbols" /></td>
<td>0 1</td>
<td><img src="image" alt="likelihood values" /></td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>300</td>
<td>(2.0)</td>
<td>0.934</td>
<td><img src="image" alt="agent symbols" /></td>
<td><img src="image" alt="event symbols" /></td>
<td>0 2</td>
<td><img src="image" alt="likelihood values" /></td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>310</td>
<td>(2.0)</td>
<td>0.980</td>
<td><img src="image" alt="agent symbols" /></td>
<td><img src="image" alt="event symbols" /></td>
<td>0 2</td>
<td><img src="image" alt="likelihood values" /></td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>320</td>
<td>(2.0)</td>
<td>0.987</td>
<td><img src="image" alt="agent symbols" /></td>
<td><img src="image" alt="event symbols" /></td>
<td>0 2</td>
<td><img src="image" alt="likelihood values" /></td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>330</td>
<td>(2.0)</td>
<td>0.987</td>
<td><img src="image" alt="agent symbols" /></td>
<td><img src="image" alt="event symbols" /></td>
<td>0 2</td>
<td><img src="image" alt="likelihood values" /></td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>340</td>
<td>(2.0)</td>
<td>0.987</td>
<td><img src="image" alt="agent symbols" /></td>
<td><img src="image" alt="event symbols" /></td>
<td>0 2</td>
<td><img src="image" alt="likelihood values" /></td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>350</td>
<td>(2.0)</td>
<td>0.987</td>
<td><img src="image" alt="agent symbols" /></td>
<td><img src="image" alt="event symbols" /></td>
<td>0 2</td>
<td><img src="image" alt="likelihood values" /></td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>360</td>
<td>(2.0)</td>
<td>0.987</td>
<td><img src="image" alt="agent symbols" /></td>
<td><img src="image" alt="event symbols" /></td>
<td>0 2</td>
<td><img src="image" alt="likelihood values" /></td>
<td>overtaking</td>
<td></td>
</tr>
<tr>
<td>370</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>380</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>390</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.18: The KERNEL results from checking for overtaking. EVENT-SO1 denotes when a switch in group element order has been identified. Note that the time sequence and agent allocation is the same as in table 5.17. Also, note that the numbers in the agent and kernel columns are buffer slot indexes.

![Diagram](image)

The relationship between buffer slot indexes and objects.
9.3 Queueing

In this grouping example we use the cluster model as a preattentive operator to represent the groups present in the data. This extends upon the mutual proximity relationship used in the DDN. In chapter 4 section 5.2 we described a queue element counting example as an illustration of how the global-form could be used. This involves two parts that need to be coordinated or planned by the global-form, which we call “queue detection”, and “count the number of queue elements”. Figure 5.33 shows some initial results that demonstrate the identification of large clusters. The problem that remains is the allocation of attentional markers and description of routines to support tasks such as counting, identifying objects that exit from the side or front of the queue.

In large groups of objects, the control could be used to guide the allocation of agents to find out information about the scene. In chapter 2 section 3.2.2 we briefly described the work of Birnbaum et al. [20] who use the causality present in the organisation of the scene components to guide the order of the scenes investigation. This approach might be applied here to guide the allocation of agents in response to the causal interrelationships of the participants.

9.4 Summary

The first grouping example demonstrates that HIVIS-WATCHER can integrate local and global reasoning to provide more complete description of identified behaviour by allocation of additional attentional resources. This incremental increase in the functionality of HIVIS-WATCHER is wholly task dependent. The queueing example indicates the need for extensions that can more fully represent and reason about larger numbers of multiple objects.
10 Conclusions

The uses for this work include identification of unusual or illegal traffic behaviour where the system could be set up by the road side and used to select when to record relevant passages of video footage for subsequent analysis. Such intelligent surveillance of a scene must be capable of recognising the situated actions of the subjects being watched and measuring these observations against some model of correct behaviour. Additional motivation for this work, and its relationship to chapter 3, comes from the theory that to learn the spatial contextual information present in SPATIAL-LAYOUT requires a task directed model of how the behaviour of objects is understood. We consider how this learning process might be performed in chapter 6, with the objective of making it easier to “set up” the HIVIS-based system for operation in new application domains.

To complete this chapter we first review the various elements of HIVIS-WATCHER developed here. Then we evaluate the results from the experiments against the surveillance problem, and finish by describing some extensions.

10.1 Elements

The main objective of this chapter has been to address the limitations of HIVIS-MONITOR we identified in chapter 4 section 5. These limitations are representative of a class of approaches we called script-based, and to solve these we adopted a more situated approach. This change in approach opened up new solutions to the surveillance problem, that were not available in the HIVIS-MONITOR framework. The central difference is the provision of timely solutions by operating in the “here-and-now”, a necessary property of the surveillance problem. This situated formulation of the surveillance problem has concerned the interpretation of the perceived motion of dynamically interacting scene objects. We have identified two elements called global-form and local-form that are needed to perform this interpretation. The global-form represents the official-observer’s egocentric “whole scene view” using one coordinate system. The local-form represents the official-observer’s application of knowledge about each perceived object to identify its exocentric field values. These two elements provide the basis for the computational theory implemented in HIVIS-WATCHER. This implementation makes use of the local frame-of-reference of selected scene objects. The local viewpoint is represented using deictic terms that simplify and decompose the spatial relationships and possible interactions between moving objects. Selecting scene objects requires attentional control that can allocate (via agent markers) processors (that run the typical-object-model) to the most task relevant objects and collect the results into a coherent description of scene activity. This control resides in the global-form and is provided by ALLOCATE and COLLECT. ALLOCATE uses a dynamic decision network (DDN), COLLECT uses task related evaluation, and together these promise a highly effective attentional control mechanism which can be distributed under the agent formalism to deliver real-time performance although in the current prototype we are interested in the competence.

As shown in figure 5.34, orthogonal to this global-form/local-form separation is the
AGENCY and KERNEL split. The AGENCY is built around the use of the typical-object-model which is part of the local-form. The KERNEL extends this functionality with a more abstract, global model of the world. The distinction between the two is perhaps exaggerated because the extension was not envisaged in the initial design (see section 1). However, the KERNEL complements the AGENCY, modelling representation and reasoning that can be performed without using detailed inspection of object properties.

The distinct nature of the local- and global-forms reflect the different perceptual frameworks and how they are used to interpret the image sequence. The CENTRAL-SYSTEM homogeneity described by Fodor [70] can still be claimed to be present, the separation between global- and local-form is more to do with levels of attentional detail.

10.2 Evaluation

The results from our investigation of the surveillance problem using HIVIS-WATCHER are much more favourable than those from HIVIS-MONITOR. We have developed a computational framework that can apply task relevant knowledge about patterns of routine behaviour to construct an understanding of what is perceived. We have demonstrated the HIVIS-WATCHER implementation, based on this computational theory, with selected scenario sequences that show its competence. These results show that HIVIS-WATCHER is very selective and focussed, which is achieved by cutting the set of peripheral triggers down to those that are relevant to the current surveillance task. However, in doing this we compromise the reaction of the system to other stimuli. This approach would not be suitable in a “survival situation” where the system’s continued existence may depend on some

---

6This orthogonal hierarchy is similar, although not intentionally based upon Brooks’ [23] task-achieving behaviour.
set of necessary triggers. In some situations this lack of completeness might be acceptable. Although the surveillance problem may not initially appear to be one of them, task bias allows HIVIS-WATCHER to ignore redundant information that HIVIS-MONITOR would spend much time processing and attend to the essential processing of its given task.

The motivation for the situated approach used here was described in chapter 2, together with relevant results from experimental psychology concerning how events and behaviour are perceived. Although these provide important background to the work described here the prototype developed here is more an engineering study rather than an implementation of identified neurophysiological findings. Within the constraints of the implementation developed here we have separated the preattentive and attentive functionality that we described in chapter 4 section 6. HIVIS-WATCHER contains task based control that organises the use of this preattentive and attentive functionality, and which makes the addition of more attentional operators, arbiters and routines largely independent of the number of objects in the scene. This is not the case for the preattentive operators, because they operate over the whole scene, making them dependent on the number of scene objects. This cost is made less significant because preattentive operators are computationally much less complex than the attentive ones.

The results from the proximity example demonstrate the advantage of using local-form agent representation and global-form task relevant control. They illustrate a solution to the spatial arrangements problem, an important problem in the context of multiple object reasoning that we found difficult to solve in chapter 4. The level of interpretation is much more detailed than that obtained in HIVIS-MONITOR, providing more timely feedback to the user. To demonstrate the general applicability of this situated approach we developed the coordination example. Unexpectedly this required additional functionality, in the form of the KERNAL, that in hindsight seems more obvious than it did during the development of this experiment. The identification of a solution using deictic representation is not always clear which may account for the rarity of its use in AI. To demonstrate how the official-observer can coordinate its attentional faculties we used the giveaway example which requires the integration of information from multiple scene objects over episode length time scales (about 5 seconds): Comparing the proximity and giveaway examples shows how changing the perceptual task alters the interpretation of visual information, rather like Yarbus' [260] experimental results. However, we are not using scanpath information, so the correspondence is more one of inspiration derived from his results.

Many of the aspects of the attentional control system outlined in this chapter have been influenced by previous work. The use of key features and markers is based on work by Agre and Chapman [3] although it was necessary to extend their techniques to include multi-agent viewpoints which contribute to the official-observer view for surveillance. The implementation has integrated this with Bayesian components that reason about the preattentive cues in a task dependent framework. The approach described here is not fully

---

7The simple answer is to add these survival related triggers to the system, but they are difficult to define and outside the scope of this dissertation.
integrated under one task selective Bayesian network, such as that proposed by Rimey and Brown [200] and implemented in their TEA program. Instead we have used the less elegant “specification of policies” to change between tasks. Although the TASKNET does integrate the results from pairs of objects and provide attentional feedback, it does not provide the flexibility described as being present in the TEA program. To more directly manipulate the temporally evolving preattentive information we have developed a dynamic decision network (DDN) that seems novel and extending it to more complex DDN graphs is challenging!

10.3 Extensions

The experiments here have shown the importance of implementing a computational theory, although we have used one implementation here, there are likely to be equally valid different approaches, some of which we described in chapter 2. The implementation developed here could be extended to address the following problems identified from the experiments described in this chapter. These extensions concern: frame rate, image-plane, causal information and more abstract control.

- We could develop a more complete queueing example by adding to the suite of KERNEL operators and associated arbiters so that HIVIS-WATCHER can perform queue related routines. However, some of the routines described in the KERNEL appear like tricks of marker allocation rather than being fragments of everyday behaviour. This may be because we do not know how a human perceiver performs these feats of understanding. The coordination and grouping examples contain careful crafted operators and arbiters that are very dependent on the chosen frame rate. This problem made the operators in the KERNEL more example specific than should really be the case. To increase the robustness and situatedness of the KERNEL operators we could use a finer grain of primitive together with faster frame rate. Robustness would be is increased by the operators being more general purpose and rather than each operator having a short sequence of process steps, each operator would just do one step enabling the CENTRAL-SYSTEM to check how appropriate each step was, before it is performed. Situatedness is increased because the system takes a more short term view, reasoning over a shorter time-span. To making each operator perform less, and yet achieve the same results means that the functionality removed from the operators needs to be replicated in the control mechanism by making it more complex.

- It would be useful to combine and extend the marker approach so that its performance has a greater correspondence to the theories of how attentional mechanisms operate. The results from such an investigation might enhance our understanding of how vision is performed, how visual routines are learnt, operate, and used. The experiments described here have all made use of the ground-plane projection. This was chosen to make depth information and interactions between scene objects more pronounced. The 3D compact encoding data also affords reasoning in the image-plane, which might be used to extend the range of operators to include those that work on image-plane data. These operators would provide a more natural link to how the data is perceived by low- and intermediate-level
vision. This extension would do more than just redisplay the results in the image-plane (an example of which is shown in figure 5.35) which in itself does not achieve anything.

- There is more causal information present in the multiple object interactions than is currently used in HIVIS-WATCHER. Using this information appears to require accessing low-level visual processing. This is illustrated by Weir's [247] AI experiments concerning fixation and data association, and Leslie's [141, 142] investigations of causal perception in infants. Also the initial work on grouping, and the gestalt primitives used here, demonstrate the need to access lower-level visual information to provide more neurologically plausible implementations.

- The agency and kernel can be considered to operate at different levels within the same framework. This could provide the foundation for a hierarchy of control, with the more abstract level operating over longer time-spans than the levels that more directly control the selected operators. These longer time-spans would support planning of observer purpose to select which behaviours to look for.

Finally, an important extension would be to apply the situated approach described here to other surveillance applications, domains and tasks. This would build upon the examples we have demonstrated which provide specific solutions to selected tasks in the road-traffic application domain.
CHAPTER 6

CONCLUSION

The aim of this dissertation has been to develop a computational theory that describes how spatial reasoning can be used to address the surveillance problem. The surveillance problem concerns more than just observation as it is task dependent, having a purpose and a goal. It is also more wide ranging than visual recognition requiring high-level visual understanding that includes knowledge about the perceived objects and the environment that they inhabit. This chapter reviews the approach we have taken to address the surveillance problem. We first discuss what was accomplished in each chapter and then identify the contributions. This leads into a description of future work and an epilogue.
Chapter 6. Conclusion

1 Recapitulation

In the first chapter we identified the surveillance problem in general terms, discussing its scope, the assumptions we would make and the properties needed in the computational theory.

In chapter 2 we identified the four layers domain, inference, task and strategic, and mapped these to areas of interest we called spatial representation, events and behaviour and control and planning. We also surveyed each of these areas identifying relevant foundations. The survey on spatial representation identified a number of potential approaches and a common representation that underlies them. The survey on events and behaviour identified and contrasted two characterisations of viable approaches we called “script-based” and “situated”. The survey on control and planning identified the benefits of both attentional control, and the traditional separation made in cognitive science between input and central systems.

In chapter 3, as a first step in addressing the surveillance problem, we identified how to represent the static domain-layer knowledge about the scene. We used the common spatial representation identified in the survey chapter to provide a model of the static spatial information that the official-observer knows about the scene. We implemented this model in a program called SPATIAL-LAYOUT.

In chapter 4 we addressed issues associated with describing dynamic spatial domain- and inference-layer knowledge. We described two formulations of the surveillance problem corresponding to the script-based and situated approaches identified in the survey chapter. These two approaches correspond to different interpretations of the surveillance problem. We identified the script-based approach first because it can be described using traditional AI techniques. The initial motivation for investigating this part of the surveillance problem was to develop a testbed to demonstrate how the spatial information held by SPATIAL-LAYOUT can be used to provide a description of dynamic object behaviour. This first formulation was implemented in a program called HIVIS-MONITOR and, although it was able to demonstrate that SPATIAL-LAYOUT provides useful results, the implementation also highlighted limitations typical of the script-based approach. To address these limitations we reformulated the surveillance problem taking a more situated perspective. In chapter 4 we began the description of the more situated interpretation of the surveillance problem. This new interpretation raised a number of control issues which we addressed in chapter 5.

In chapter 5 we addressed task and strategic-layer control issues raised by the situated approach. We developed a dynamic control mechanism that used preattentive and attentive spatial reasoning to identify and reason about objects that are likely to be relevant to the current surveillance task. The HIVIS-WATCHER program addresses the limitations found in HIVIS-MONITOR and, although it more fully addresses the surveillance problem, HIVIS-WATCHER does not solve it completely. Surveillance is a difficult problem and in this dissertation we have only really identified this as an area that requires further research.
All the main programs described here (e.g., SPATIAL-LAYOUT, HIVIS-MONITOR and HIVIS-WATCHER) have been implemented by the author in Common Lisp and run on a Sun-4. Only the MAP-EDITOR described in chapter 3 section 3.1 was developed by other members of the VIEWS project. None of the code developed here was included in the final VIEWS project implementation.

HIVIS-MONITOR and HIVIS-WATCHER have been tested on the sequences shown in appendix A, with their main functionality illustrated by the examples given in chapters 4 and 5.
CHAPTER 6. CONCLUSION

2 Contributions

2.1 Spatial representation

We developed the cellular topology to support the topological and metrical relationships we identified as being required in the surveillance problem. These requirements arose from the objective of implementing a database to hold the a priori knowledge about the scene which includes ground-plane geometry and the semantic information that is attached to it. This knowledge is intended to represent the general background information that someone who knows the scene would have, including things like a model of the typical behaviour exhibited by the objects that occupy (or use) this space.

The advantages offered by the cellular topology arise from its mathematical foundation, and we use this to provide a separation between implementation and representation. The benefit of having the ground-plane data independent from its implementation, is that the implementation can be changed without greatly affecting how the data is used. This abstraction is provided by the mathematical definition of how cells structure space. This means that we can define regions in terms of cells and not have to know how the cells themselves are implemented. All we require is the mathematical definition of cells and how they can be used.

In addition to this, we use two types of region. Leaf regions provide the elementary spatial units used to describe the ground-plane (i.e., they structure the space of the ground-plane). We do not attach properties to leaf regions, for this we use composite regions. Each composite region expresses the spatial extent, in terms of connected leaf regions, of a particular property value. If we think of leaf regions as being the leaves in a tree-like data structure then the composite regions are the nodes in this tree to which the property values are attached. These property values are accessed via an object’s occupancy of a leaf region. By following the parent links up from the identified leaf region the various composite region property values can be obtained to provide contextual indexing of the object. Although the mechanism is general, the data obtained is specific to the application domain for which the spatial-layout database has been constructed.

We attach semantic properties to the composite regions denoting what typically happens in that region of space. Contextual indexing accesses this information, obtaining a semantic description as a set of values which provide a sketch of the typical behaviour that the object might be engaged in. This information about typical behaviour is used to constrain the interpretation.

The contributions concerning spatial representation are:

- The use of cellular topology to provide a foundation and abstraction that supports different implementations.
- The separation of the representation into spatial extent and spatial attributes for the attachment of behavioural semantics.
- Contextual indexing, which is the provision of spatial information (behavioral and physical) based on the place of an object in the scene. Although obvious, this does not
appear to have been addressed before. We have also shown that contextual indexing scales well with scene complexity.

2.2 Events and behaviour

We developed a computational approach that uses the spatio-temporal features resulting from model-based tracking to identify the object events and behaviours present in this signal. These behavioural descriptions provide a more detailed understanding of the data present in the original image sequence, with the objective of providing its conceptual description. HIVIS-MONITOR was developed to illustrate the competence of contextual indexing as an effective first step towards the formation of conceptual descriptions of object behaviour. The implementation of HIVIS-MONITOR is able to identify region crossings and kinematic events by using a generic approach that we illustrated with domain specific event composition rules using cellwise-time to get episodes. The results show that contextual indexing can be used to obtain partial conceptual descriptions. The implementation also demonstrated the feasibility of this approach to some applications and the difficulty of describing the complex dynamic interactions exhibited by the objects. This failure to describe complex interactions is particularly apparent in the spatial arrangements problem which involves describing the spatial relationship of two objects. This problem was unexpectedly hard in the ground-plane's coordinate system. To more fully capture this dynamic spatial information we identified the usefulness of employing the local viewpoints of the individual participants and indexing their local environment in their own deictic terms.

This switch from the global view of the ground-plane representation to the local object centred approach introduced the need for composition rules representing how the official-observer combines the results of the deictically referenced object aspects. The development of this mechanism to coordinate the local viewpoints of the agents identifies more complex control issues.

During the work on events and behaviour the reformulation of the surveillance problem changed the objectives of this thesis. We found that the initial objective of extending SPATIAL-LAYOUT to overcome the problem of providing the a priori knowledge about typical object behaviour, required a better model of object behaviour. The development of this behavioural model itself uncovered problems relating to control and the surveillance task. These all need to be solved before returning to the problem of learning the behavioural and environmental spatial properties associated with the scene. The approach resulting from the reformulation builds on Agre's [2] identification that the deictic approach provides an opportunity for integrating vision research, AI and social science.

The contributions concerning events and behaviour are:

- The distinction between script-based and situated approaches.
- The separation and integration of global and local reasoning in the context of a single official-observer. The illustration of how both play complementary roles in developing different levels of understanding.
2.3 Control and Planning

Control issues were not encountered in HIVIS-MONITOR because it uses a simple pipelined architecture. In our third program, called HIVIS-WATCHER, we rejected this pipelined approach, instead adopting a more closed-loop approach which raised a number of control problems. The investigation of control issues may seem far removed from spatial reasoning, however, our examination of the problems highlighted by the deictic approach demonstrated the important role control plays in the interpretation of spatial information. To make this local reasoning viable the control mechanism uses a preattentive and attentive processing split. The attentive processing concerns the local deictic reasoning and the preattentive processing selects the objects on which to perform the attentive processing. The preattentive processing uses a dynamic decision network (DDN) to reflect the dynamic scene structure as described by the task related preattentive operators. The DDN is used to hold information about which objects are worth attending. Once objects have been identified as likely to be task related, the more detailed attentional operators are run and their results are combined. For some tasks integrating results is a tricky process because of the deictic references employed and the requirement of temporal continuity to make sense of the index related results. To solve this problem we use a further Bayesian network which feeds back its result from the evidence collected so far to the DDN for it to gauge whether to continue or terminate attention of the objects involved.

The traditional separation made in cognitive science between input and central systems provides a description of the two tightly coupled components in HIVIS-WATCHER. The input system holds the functionality that obtains object aspects, while the central system guides which objects and aspects should be obtained to fulfill the given surveillance task.

The separation of preattentive and attentive processing, and the use of a task directed control mechanism provides a focus-of-attention that only processes what is needed for the task related features that the official-observer is looking for. This addresses the problem of selective attention, by using high-level data association. HIVIS-WATCHER provides timely surveillance information about what is happening in the scene.

The contributions concerning control and planning are:

- The use of high-level selective attention to allocate deictic processing and collect the results from this processing.
- The extension of an egocentric agent model to the exocentric interpretation of multiple agents.
- The propagation of reasoning in the “here-and-now” through to the control mechanism in order to reflect the reactive quality of dynamic object behaviour.
3 Future work

The areas of future work mostly stem from the assumptions made in chapter 1.

- **For assumption 1** The computer programs are just prototypes that are not yet required to operate in real world applications. Extending the HIVIS approach for applications that operate in the real world is an important objective. This may also include a model of language use (such as that described in Chapman's [34] SONJA system) to allow the human operator to interact with the surveillance system at runtime.

- **For assumption 2** We are using a known 2D ground-plane. Extending the surveillance system for operation in new scenes for which it does not have prior knowledge is likely to be of increasing importance as advanced vision systems are developed for commercial applications. We briefly consider how this might be accomplished in section 3.1.

- **For assumption 3** The ground-plane is static. This is a different facet of learning properties about the environment where the surveillance system would need to adapt to changes in the background scene, and typical object activity. For example, in the road traffic domain, such a change could be caused by roadworks or road alterations or a car breaking down. We briefly consider this problem in appendix C section 2.2.5.

- **For assumption 4** We are given a stream of compact-encodings that represents the output from intermediate-level processing. This extension is to remove the separation introduced in the VIEWS project (see chapter 1) between perceptual and conceptual processing. This would require the integration of HIVIS with low- and intermediate-level vision in a closed-loop system.

- **For assumption 5** The official-observer has a viewpoint from a single fixed camera. Research concerning active vision (see Ballard [14]) is addressing this problem, including issues associated with a moving official-observer.

- **For assumption 6** The surveillance task is understanding the activity of moving objects in the scene. Extending the surveillance task to include participation in interactions and so removing the detached quality of the observer.

- **For assumption 7** Full temporal reasoning is not required. Include a better notion of time for purpose of planning and applying previous experience.

- **For assumption 8** The description of system behaviour and the behaviour of observed objects can be defined by the program designer. Obtain knowledge about the participants in the scene via learning and situated experience.

These are all difficult problems, which is why we made the assumptions in the first place. These problems can be broadly separated into: increasing the active participation of the official-observer; and learning environmental properties about the scene and fellow participants. This extends further the active vision approach in recognition of the problems connected with understanding what we see.
3.1 Learning intrinsic environmental properties

When we attend to objects in or moving through our field-of-view, we bring to our understanding of what we see a great deal of visual knowledge. Some of this knowledge is particular to the scene (or place) we are observing, whilst other information is of a more general scene independent nature. If we watch the scene for a while we can build up a framework of expectations, describing what typically happens and where it occurs. This information about activities enriches our scene dependent knowledge allowing for a more complete understanding of what is happening in a scene. The main question this future work is designed to answer concerns the way in which this scene dependent knowledge is acquired. Can it be done just by watching what happens in a scene, and if so, how does it work?

To facilitate the identification of typical behaviour it is necessary to select environments where behaviour is exhibited in the form of identifiable and structured "routines". There needs to be some level of familiarity with what is to be learnt for the new data to be put in context. In these domains, we can expect the identification of exhibited object behaviour and the places in which it occurs to help form an understanding of what is taking place. We are not directly concerned with how we might represent and acquire geographic knowledge (Davis' MERCATOR system [49] provides an example of this approach), instead the identification and construction of typical behaviours and spatial context is closer to the work of Ballard [14] and Whitehead [252]. The key point is that vision may be more readily understood in the context of the visual behaviours that are engaging the official-observer.

We need to consider how observing similar scenes or the same one from different view points can improve the method by which visual knowledge is learnt. We do not need to directly address the problem of moving through an environment (i.e., a dynamic perceiver), but it would be of great benefit if we could identify relationships and invariants between previous knowledge of different locations and places, and how learning of this new location is performed. For example, is it necessary to learn a complete 3D map of the world, or will some other less complete representation suffice?

The process of building up the scene knowledge can be called learning intrinsic-environmental-properties. Previous work that comes closest to this notion of intrinsic-environmental-properties is Gibson's affordances [85], for example, if the ground is solid then the surface "affords" support. However, this affordance is dependent upon the animal involved and the task to which the surface is being put. Intrinsic-environmental-properties capture information about what typically happens in the scene. They build up a framework of expectations which we can use to help describe what a scene object is doing. These intrinsic properties are based upon object features such as direction-of-motion, or location in the scene.

To make the learning process more feasible we could use HIVIS-WATCHER's task-related approach. This would allow the learning system to perform a predetermined attentional task, for example, in a road-traffic scene including a junction we might say identify where vehicles give way to each other. When performing this visual task, all unrelated
object actions should be ignored so that the system focuses on those objects that "look as though" they are involved in what the official-observer's typical-object-model says is giveaway-behaviour. By making the acquisition process task-related we reduce the burden of learning everything at once, providing a more computationally feasible "biased-learning" approach.

The requirement for work on visual acquisition became very apparent when, on the VIEWS project, we had to input all the a priori geometric and behavioural information for a given scenario for the visual surveillance task by hand. This time consuming task would need to be performed for each new scene. This future work outlines a new area of research that is designed to overcome this knowledge acquisition problem by making use of scene independent knowledge concerning: general information about the scene (affordances of the environment), and task knowledge about the dynamic scene objects and how these are perceived by the official-observer. The objective of this extension is to address the issue of perceptual learning from the standpoint of how "intrinsic-environmental-properties" are learnt. To do this we have separated scene knowledge into two classes to investigate how the scene dependent knowledge can be acquired, and what level of scene independent knowledge is necessary to perform this process.

3.2 ALTERNATIVES TO REPRESENTING THE WORLD

It would be useful to determine whether the knowledge heavy approach used in the HIVIS-systems is appropriate, or whether more active systems are appropriate. This future work is intended to increase the active participation of the official-observer, and make its participation in the environment provide the input it requires to learn about how surveillance (and other perceptual tasks) are performed. This not only includes knowledge about the scene and the other participants, but also the skill of performing the perceptual task in relation its current high-level goal (see Ballard [14]). An objective would be to reduce the amount of prior knowledge, and yet retain enough to exploit the implicit information in explicit environmental cues (e.g., road markings, moving objects).

3.3 A UNIFIED MODEL

In addition to applying cellular topology to vision research, Fleck [68] also proposes its use in a number of areas related to AI. Here we have tried to extend the application of cellular topology, and it may be profitable to develop a unified model using this mathematical foundation, and extend its use to other domains. Unifying vision and natural language to communicate results might be a first step in this direction, which could make use of Fleck's natural language semantics defined using cellular topology. Communication is a two way process, so we would not want to adopt the query-based, off-line solution of HIVIS-MONITOR. HIVIS-WATCHER provides a better solution, however, the lack of a true closed loop to intermediate-level visual processing might need addressing. A more complete system is described by Chapman [34] who uses "instruction use" to support situated communication between the user and the machine based vision system.
4 Epilogue

This work represents some steps toward the objective of deriving conceptual descriptions corresponding to what the official-observer currently perceives. There is a lot more to be done in the investigation of how surveillance is performed. Some of these problems are due to the lack of knowledge about how we perform the everyday task of understanding the activity of other participants in the world. Other problems are due to the gap between the real world and formal models of it. Developing implementations makes this gap more apparent because of the necessary assumptions made just to begin the investigation of the problem.

In the HIVIS-based systems we have developed AI approaches that recognise the importance of the continuously changing world. To make this tractable we have used a static camera model of the perceiver who does not participate with the environment. Even so, we have extended computational vision research beyond the recognition of objects. Often it seems that vision research is assumed to just concern static images and the recognition of objects, and that AI research can just assume that necessary information about the world can be obtained by intermittently using a visual sensor when visual input is required. Here we have tried to incorporate the dynamics present in visual data. Although this research has been mainly illustrated by using data from a road-traffic surveillance application, the intention was that the general framework should be applicable to other application domains.
APPENDIX A

VIDEO FOOTAGE AND SCENARIOS

Currently, three sequences have been produced from video footage collected at the German roundabout. The sequences are 27, 55 and 109 frames in length. Both HIVIS-programs work on all three sequences.

<table>
<thead>
<tr>
<th>scenario</th>
<th>sequence</th>
<th>number of frames</th>
<th>pages</th>
<th>figures</th>
</tr>
</thead>
<tbody>
<tr>
<td>occlusion</td>
<td>1</td>
<td>27</td>
<td>2</td>
<td>A.1, A.2</td>
</tr>
<tr>
<td>enter</td>
<td>2</td>
<td>55</td>
<td>3</td>
<td>A.3, A.4, A.5</td>
</tr>
<tr>
<td>queueing</td>
<td>3</td>
<td>107</td>
<td>3</td>
<td>A.6, A.7, A.8</td>
</tr>
</tbody>
</table>

The occlusion scenario contains two episodes running in parallel. As the car overtakes a lorry, they both traverse the roundabout, causing a car waiting to enter-the-roundabout to give way to them.

The enter scenario\(^1\) begins with a car giving way to a tram (not represented because the intermediate-level vision model-matcher did not have a model for trams). This is followed by an overtaking episode and an enter-the-roundabout episode.

The queueing scenario begins with an enter-the-roundabout episode (with near collision, or model-matcher position error). This is followed by the formation of a queue that is giving way to an unseen tram. Once the queue is formed vehicles leave and join it. Finally the queue begins to dissolve.

The occlusion, enter and queueing scenarios contain all the posebox data from the Bremer Stern roundabout that was provided by the Perception Component (see chapter 1 section 2). Some additional scenarios were created from “cut-ups” (where isolated object paths from different scenarios are combined to create new ones). For example, the following scenario given in figure A.9 was used to test the reallocation of markers via the mutual-proximity test, and the early exit scenarios (see figures A.10, A.11 and A.12) were used to test the detection of give way episodes, identify the problems described in chapter 5 section 8.2.3 and help develop the grouping preattentive cue.

In hindsight the development of a discrete event simulator would have solved the shortage of test data and facilitated better empirical evaluation.

\(^1\)To fit the enter scenario sequence on three pages, frame 90 has been removed.
Figure A.1: The occlusion scenario (part 1).
Figure A.2: The occlusion scenario (part 2).
Figure A.3: The enter scenario (part 1).
Figure A.4: The enter scenario (part 2).
Figure A.5: The enter scenario (part 3).
Figure A.6: The queueing scenario (part 1).
Figure A.7: The queueing scenario (part 2).
Figure A.8: The queueing scenario (part 3).
Figure A.9: The following scenario.

Figure A.10: The early exit scenario (version 1).
Figure A.11: The early exit scenario (version 2).

Figure A.12: The early exit scenario (version 3).
APPENDIX B

MATHEMATICAL DEVELOPMENT

In this appendix we present the topological foundations for the common spatial representation identified in chapter 2, describe how these are used to represent regions, provide further implementation details, and briefly discuss tolerance space.

We begin with the background for the common spatial representation, called "cellular topology", which we place in a historical context, introducing the topological terms needed to express it. The central idea of cellular topology is expressed in definition B.13. In the spatial representation described here we are only considering metric topologies, using them to describe "continuity reasoning" and "connectivity". We model continuity reasoning as a function that has a constant value over a closed space and we use this to represent the extent of a given property value. These spatial extents are used to determine the structure of space that is expressed by the leaf region tiling of the ground-plane, and the extents of the composite regions that denote these property values in SPATIAL- LAYOUT.

We continue with background material concerning the topological terms used to describe and manipulate regions. This is followed by a discussion of tessellation techniques, covering why some of the standard approaches are not appropriate for describing the regions in SPATIAL-LAYOUT, pointing out some efficient algorithms, and describing interesting approaches to optimal decomposition both in terms of the number of polygons and the fiducial size of the cells produced. Continuing the theme of implementation we describe the database primitives used in SPATIAL-LAYOUT (see chapter 3), provide an example of a better format for the spatial layout data file, and describe the "CELL-PARSER" used to generate the close-edge model in the regular square cell implementation.

This appendix finishes with a description of tolerance space which we identified in chapter 4 section 1.1.2 as providing a way of modelling the increase in noise due to distance from the camera.
1 Background for the Common Representation

Instead of trying to generalise all the spatial representations to obtain the common form, we will consider the underlying mathematics. Mathematicians usually divide spatial concepts into two categories only: metric and qualitative. In the qualitative category we have topology which takes no account of distance. And in the metric category we have the concept of distance, such in Euclidean geometry where we have the concepts of length and angle, to which we can add a Cartesian system to give a familiar representation of space. Adding this metric removes the property of allowing arbitrary smooth deformations ('fixing' the topological order, if you like), although we make use of other topological properties like: proximity, separation, order, surrounding, continuity, connectivity, enclosure. Indeed, one of the most important ways of imposing a topology on a set is to use a metric.

Definition B.1 A metric on a set $X$ is a function $d : X \times X \rightarrow \mathbb{R}$ such that the following are true:

1. $\forall x \in X \quad d(x, x) = 0$
2. $\forall x \in X \quad (\forall y \in X) \quad d(x, y) \geq 0$
3. $\forall x \in X \quad (\forall y \in X) \quad d(x, y) = d(y, x)$
4. $\forall x \in X \quad (\forall y \in X) \quad (\forall z \in X) \quad d(x, z) \leq d(x, y) + d(y, z)$

Given a metric $d$ on $X$, the number $d(x, y)$ is often called the distance between $x$ and $y$ in the metric $d$, and $X$ together with $d$ is called a metric space $(X, d)$.

The Euclidean distance can be defined as $d(a, b) = [(a_1 - b_1)^2 + \cdots + (a_n - b_n)^2]^{\frac{1}{2}}$.

The subject of topology seems less well known in comparison to the everyday form of Euclidean geometry. To compensate for this, let us briefly cover some topological concepts and historical background that will assist our investigation of spatial representations.

1.1 Topology

Topology is a branch of mathematics that deals with selected properties of related physical or abstract elements. The properties are those that remain invariant when a collection of elements undergoes distortion, as long as it remains intact. The distance between points may change, but their positional order stays the same. Despite its basic character, topology did not reach its present level of development until the 20th century, although its roots lie in the 19th century researches of mathematicians such as Bernhard Riemann, Georg Cantor and Henri Poincaré (see for example Lefschetz [140]).

We are interested in the subfield of algebraic topology (for more details see Munkres [168] or Spanier [218]) which deals with the algebraic manipulation of symbols that represent geometric configurations and their relationships. The fundamental methodology of algebraic topology can be summarised as a three step algorithm:

1. Convert the problem from a geometric statement about geometric configurations, to an algebraic statement about symbols.
2. Solve the algebraic form of the problem.
3. Convert the solution back into a geometric statement.

In order for us to describe the topological constructs used to represent space we need to introduce some basic topology. We begin with point-sets, which have a more intuitive feel than other forms of topology. Point-set topological spaces (for more details see Munkres [167] or Flegg [69]) are based on the concept of “open” sets. A set is said to be open in a metric space if every point in the set is surrounded by points also in the set, for some slight distance.

**Definition B.2** An open set is a set $A$ such that $(\forall e \in A)(\exists e \in \mathbb{R}, e > 0)(\forall x \in X)((d(x, a) < e) \Rightarrow (x \in A))$.

**Definition B.3** A topology on a set $X$ is a collection $T$ of subsets of $X$ having the following properties:
1. $\emptyset$ and $X$ are in $T$, i.e., $(\emptyset \in T) \land (X \in T)$.
2. The union of the elements of any subcollection of $T$ is in $T$, i.e., $(\forall S \subset T,(\cup S) \in T)$.
3. The intersection of the elements of any finite subcollection of $T$ is in $T$, i.e., $(\forall A, B \in T,(A \cap B \in T))$.

A set $X$ for which a topology $T$ has been specified is called a topological space and is denoted by an ordered pair $(X, T)$.

A “topological space” then removes the concept of “distance” and relies only upon the concept of “open set”.

It is quite easy to demonstrate that the above definition of open in metric spaces yields a topology. Such a topological space is called metrizable however, not all metrics produce a topology due to the requirement of regular second countability [221]. Although point-set topology is useful for analysis and obtaining an intuitive feel of what is going on, we need to use algebraic topology.

In algebraic topology, the geometric space is divided into sets, and the topological relations between these sets are reduced to algebraic equations. By doing this we no longer need to manipulate large numbers of coordinates that represent geometric objects, instead we can manipulate symbols using a simple algebra. This algebra is based upon the primitives closure, interior and boundary, together with the standard set theory primitives union, intersection and difference. The following definitions of these primitives include both an algebraic and point-set definition. The algebraic definition is given first with the point-set definition used to provide a more intuitive description.

**Definition B.4** The closure of a set $A$ is the set together with its limit points, denoted by $\overline{A}$ or $A^{-}$. Since a set which contains its limit points is closed, the closure of a set may be defined equivalently as the smallest closed set containing $A$. In point-set topology a closure is the collection of points within the “universal” set that are “close to” the set, i.e., the closure $X$ of $A$ is $\{x \in X | (\forall U \in T)(x \in U) \Rightarrow (U \cap A \neq \emptyset)\}$.
Definition B.5 The interior of a set \( A \), denoted \( \overset{\circ}{A} \) or \( A^o \), is the largest open set contained in \( A \), or equivalently, the union of all open sets in \( A \). In point-set topology an interior is the collection of points that are completely surrounded by the set, i.e., the interior \( X \) of \( A \) is \( \{ x \in X \mid (\forall U \in T)(x \in U) \land (U \subset A) \} \).

Definition B.6 The boundary of \( A \), denoted \( \partial A \) is the set of all points which are in the closure of \( A \) but not in the interior of \( A \). In point-set topology a boundary is the set of points within the universal set that are “close to” both the set and its complement, i.e., the boundary \( X \) of \( A \) is \( \{ x \in X \mid (\forall U \in T)(x \in U) \Rightarrow (U \cap A \neq \emptyset) \land (U \cap (X - A) \neq \emptyset) \} \).

We can use this simple algebra to define relationships like: (1) \( \overset{\circ}{\partial A} = \overset{\circ}{A} \cup \partial A \), (2) \( \partial A = A \cup \overset{\circ}{A} \), and (3) \( \overset{\circ}{A} \cap \partial A = \emptyset \). We will also require the concepts of separation and connectedness.

Definition B.7 If two sets \( A \) and \( B \) have the property that \( \overset{\circ}{A} \cap \overset{\circ}{B} = A \cap \overset{\circ}{B} = \emptyset \), they are called separated. A set \( A \) in a topological space \( X \) is connected if it cannot be written as the union of two separated sets.

Having defined the primitives used in algebraic topology, we are now able to describe two different approaches to representing objects, the first is called “simplicial topology” and the second is called “cellular topology”.

1.2 Simplicial Topology

One of the basic tools of topology is the “simplex” which was developed by Poincaré providing what is called simplicial or “combinatorial” topology (Alexandroff [4, pages 11–12]). An \( n \)-dimensional simplex or \( n \)-simplex is the convex hull of \( (n+1) \) points embedded in a space of dimension \( n \) or greater. Thus a 0-simplex is a single point, a 1-simplex is a line joining two distinct points and a 2-simplex is a filled triangle joining three non-collinear points. Assuming the \( n \)-simplex is defined by an ordered set of \( n+1 \) distinct points \( a_0, a_1, \ldots, a_n \) we can write the simplex as \( < a_0, a_1, \ldots, a_n > \). The ordering of the points within a simplex is considered to be the orientation, so that exchanging any two vertices in the list is said to reverse the orientation (e.g., \( < \ldots, a, \ldots, b, \ldots > = -< \ldots, b, \ldots, a, \ldots > \)).

The boundary of an \( n \)-simplex consists of the union of the \( (n-1) \) simplices, called faces. The boundary of a 2-simplex \( < a, b, c > \) is made up of the three lines \( < a, b >, < b, c > \) and \( < c, a > \). In simplicial topology [4] the boundary operator works on a vector space (over integers or some other algebraic field) generated by using the “\(< \cdot >\)” symbol for each simplex and using orientation reversal as negation. The boundary of a 0-simplex is the zero-vector having only one orientation (i.e., \( \partial < a > = 0 \)). The boundary of a 1-simplex is the vector difference of its two 0-simplices \( \partial < a, b > = < b > - < a > \). The boundary of a 2-simplex is the closed sum of its oriented edges \( \partial < a, b, c > = < a, b > + < b, c > + < c, a > \).

Definition B.8 ([4, page 20]) Let \( C^n \) be a complex with \( C^n = \sum t^i \alpha^n_i \), and \( x^n \) be an oriented simplex \( x^n = < a_0, \ldots, a_n > \). The boundary \( \partial C^n \) of the complex \( C^n \) is the algebraic sum of the boundaries of the oriented simplices \( x^n_i \), i.e., \( \sum t^i \partial x^n_i \), where
(1) the boundary of the oriented simplex \( x^n \) is the \((n-1)\)-simplicial complex, and (2) \( f^i = +1, -1 \) or 0 according to whether the \((n-1)\)-simplicial complex occurs in the boundary with coefficient +1, -1 or not at all. The general formula is \( \partial \{ a_0, a_1, \ldots, a_n \} = \sum_{i=0}^{n} (-1)^{i} \{ a_0, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n \} \) which sums over the extrema points ignoring each of them, one at a time.

This boundary operator interacts with unions of simplices. Assuming we have a set \( S \) of simplices, then the boundary of the union of these simplices can be written as an algebraic sum of each boundary of the component simplices as follows: \( \partial(\bigcup S) = \sum_{s \in \partial S} \partial s \). Note that the shared oriented edges cancel.

This topological structure provides a good approach for representing polyhedra. However, it does require a triangulation which might not always provide the most appropriate shape primitive.

1.3 Cellular Topology

Cellular topology was developed by J.H.C. Whitehead [251] providing a more complicated notion of topology that is similar to the simplicial complex. The spatial representation used in this thesis is based upon the cellular topology, and here we are defining the background necessary to describe what a particular form of cellular topology, called a CW-complex, is. The CW prefix is an acronym of “closure-finiteness” and “weak topology” and highlights their importance. Cellular topology uses homomorphisms. A homomorphism (meaning “of similar form”) between two geometric objects is an invertible function from one to the other such that both the function and its inverse are continuous. This essentially means that there is an identification of the points in the source object to the points in the target object that preserves the concept of “near”, and introduces no new tears, holes or seams. The primitive form in cellular topology is the n-ball.

**Definition B.9** An \( n \)-ball is the set of points in Euclidean n-space, \( \mathbb{R}^n \), with distance from the origin less than or equal to one. An \((n+1)\)-ball is \( B^{n+1} = \{ s = (s_0, s_1, \ldots, s_n) \mid |s| \leq 1 \} \). The surface of an \((n+1)\)-ball, which consists of all the points at “distance from the origin” equal to 1, is called a \( n \)-sphere, \( \partial B^{n+1} = S^n = \{ s \in B^{n+1} \mid |s| = 1 \} \), and the difference between a \((n+1)\)-ball and a \( n \)-sphere, called the open ball, is \( B^{n+1} \setminus S^n \).

We do not deal with \( n \)-balls and \( n \)-spheres directly, these act as topological ideals, instead we use \( n \)-cells.

**Definition B.10** An \( n \)-cell is any object homeomorphic to an \( n \)-ball (this include an \( n \)-simplex). The boundary of an \( n \)-cell is that portion of the \( n \)-cell mapped onto the \((n-1)\)-sphere by any homeomorphism.

A \( n \)-dimensional cellular complex (\( n \)-complex) is a collection of \( n \)-cells, such that portions of their boundary (homeomorphic to \((n-1)\)-cells) have been identified to one another. The
Figure B.1: Representing a sphere in simplicial and cellular topology.

complete algebraic machinery from simplicial algebraic topology also exists in cellular algebraic topology, with appropriate interpretation of cells as vectors and orientation.

For a 2D representation, the cells are made up of three cell-dimensions: 0-cells which describe vertices (or points); 1-cells which describe edges (or lines); 2-cells which describe faces (or surfaces or planes). If we extend this to a 3D spatial representation, then we also have 3-cell solids.

The big difference between simplices and cells is for visualisation and storage. A geometric configuration of the type we are concerned with is called a topological manifold. A cellular realisation of a topological manifold is a cellular complex that is homeomorphic to the manifold. Normally a cellular realisation of a topological manifold requires fewer cells than a simplicial realisation. For example, as shown in figure B.1, a solid sphere needs one 3-cell, two 2-cells (the northern and southern hemispheres), one 1-cell (the equator) and one 0-cell (where the ends of the equator meet), which gives six cells in total. For a simplicial realisation it would require one 3-simplex, four 2-simplices, six 1-simplices and four 0-simplices, which is fifteen in total, to describe the 3D oriented simplex of a tetrahedron, providing a topological equivalent to our sphere.

We can use our n-cells to define the structure of a particular space, by using regular cell complexes (for details see Munkres [168, pages 214-221] and [152, pages 76-104]). We first need to stipulate the separation axiom of our topological space by using a definition of neighbourhood and Hausdorff.
Definition B.11 If \( x \) is a point of \( X \), and (open) **neighbourhood** of \( x \) is an open set that contains \( x \).

**Definition B.12** A space \( X \) is **Hausdorff** if given two distinct points \( x_1, x_2 \in X \), there exist neighbourhoods \( U_1, U_2 \) of \( x_1, x_2 \) respectively, such that \( U_1 \cap U_2 = \emptyset \).

Now we are in a position to define what a regular cell complex is. This is the essential class of model that we are using in this work.

**Definition B.13** ([168, page 214]) A **regular cell complex** or **CW-complex** is a space \( X \) and a collection of disjoint open cells \( e_\alpha \) whose union is \( X \) such that:

1. \( X \) is Hausdorff.
2. **(Closure-finiteness)** For each \( n \)-cell \( e_\alpha \) of the collection, there exists a continuous map \( f_\alpha : B^n \to X \) that maps \( \tilde{B}^n \) homeomorphically onto \( e_\alpha \) and carries \( \tilde{B}^n \) into a finite union of open cells, each of dimension less than \( n \).
3. **(Weak topology)** A set \( A \) is closed in \( X \) if \( A \cap \tilde{e}_\alpha \) is closed in \( \tilde{e}_\alpha \) for each \( \alpha \).

Intuitively, a regular cell complex complex is a space which can be considered as a union of disjoint "open cells" providing a tiling of space. We have now covered the material necessary to describe Fleck's extension to the cellular topology.

### 1.4 Adjacency Sets and Incidence Structure

Fleck's work provides a proof that a structure-preserving mapping between the adjacency structures of two cell complexes determines a homeomorphism between their underlying spaces. This result provides a simpler route for showing that two cell complexes are homeomorphic by comparing their adjacency structures rather than building homeomorphisms directly. This structure-preserving mapping is used to fuse stereo image pairs.

Adjacency sets were developed by Fleck in [67] and given with proofs in [68], and are used to specify the adjacency relationships between cells.

**Definition B.14** (Lemma 5 [68, page 389]) An **adjacency set** must comply with the following:

1. Every \((n-1)\)-cell must be a face of at least two \( n \)-cells.
2. There is a fixed dimension \( n \), such that each cell is either an \( n \)-cell itself or it is a lower dimensional face of an \( n \)-cell.
3. The intersection of any set of cells must be exactly one cell or empty.

In this definition, the first clause ensures that neighbouring regions have a common edge and that an edge has two vertices. The second clause ensures that space is of a constant maximal dimension. The third clause ensures that two neighbouring regions can only share one edge. An example set of cells is given in figure B.2. A, B, C, and D are 2-cells. A is adjacent to B along a 1-cell edge called \( \{A, B\} \). A, B, C, and D all share a common vertex \( \{A, B, C, D\} \).
Another approach to naming the boundary can be based on a different way of considering the structure of space. This approach is more general than the adjacency sets but a little bit more unwieldy.

**Definition B.15 ([68, page 384])** The incidence structure of a regular cell complex $X$ consists of a list of all cells in $X$ together with an incidence relation $\text{Face}$ on this set of cells such that $\text{Face}(x, y)$, if and only if, $x$ is a face of $y$.

For the purposes of the current example (and how this representation can be implemented), consider the vertices as coordinates. This gives each vertex a unique name. Each vertex is the Face of at least two edges (by Lemma 5) and, by the construction of edges, every edge-cell can be named by its two vertices. We can also extend the constructivist approach to naming a 2-cell by using a set of its edge-cells. By constructing this hierarchy of names, we have introduced naming redundancy. This redundancy can be removed in the implementation by using a winged-edge boundary representation [102]. This model can be used to describe a closed region.

In either case, we end up with a set of cell addresses.

### 1.5 Inter Region Boundaries

It is not always easy to describe the boundary between two objects. Often we have the question of which object owns the boundary? For example consider figure B.3 does the boundary between the table and the mug belong to the mug, the table, both, or neither? Sometimes the boundary is arbitrarily assigned to one or other of the objects or, in the case where one of the objects is the background, boundary ownership might be treated differently. Alternatively we might have the boundaries overlap by one point so that both objects own the boundary. None of these approaches provide a good representation of the boundary between two objects.
Figure B.3: A mug on a table.

Figure B.4: An example of a rectangular space tessellated into two triangles and four models using various forms of square cells. (a) shows an open-edge boundary with the deleted cells shaded; (b) is a better implementation of the open-edge model; (c) shows the initial implementation that required allocating the cells the make up the boundary to one region or the other; (d) the closed-edge model.

In her description of cellular topology, Fleck [68] describes two basically equivalent ways of solving this problem, which are developments of the two boundary models described above. First of all we extend the definition of regular cell complex to include boundaries.

**Definition B.16** ([68, page 394]) A regular cell complex with boundaries is a regular cell complex together with a list of cells called the boundaries of the complex.

When reasoning about cell complexes we are primarily interested in this definition of boundaries. We can use either an open-edge or a closed-edge boundary model throughout our spatial representation. In the open-edge model of boundaries, points corresponding to boundary adjacency sets are simply deleted from space. This means that the cells to either side of the boundary are now next to each other but no longer connected. The closed-edge boundary model is similar but here points are added to “close” the edges, making them look like closed subsets of real space. The new points on either side of the boundary are right next to each other but distinct. See figure B.4.

This extended cellular topology solves the **boundary ownership problem**, although care needs to be taken when implementing the cell descriptions. Take note of the cell adjacency rules, which when followed provide sensible neighbour relationships as part of the cellular description of the connectivity of space. We also add a Euclidean metric on top of the basic cellular topology to provide a coordinate system, which is needed because in addition to qualitative reasoning, quantitative reasoning is also used in our applications.
2 Background for Regions

Here we provide formal definition that define a region in terms of being a subcomplex, to show (1) how regions can both be used to tile a planar regular cell complex, and (2) that regions can also be constructed in terms of other regions. We begin by introducing the concept of skeleton.

2.1 The Skeleton

**Definition B.17** ([152, page 80]) The $n$-skeleton of $X$, written $X^n$, is the regular cell complex consisting of all cells of $X$ whose dimension is less than or equal to $n$.

In terms of $n$-skeletons we can say that for all $n \in \mathbb{N}$, the $n$-skeleton of $A$ is the intersection of $A$ with the $n$-skeleton of $X$: $A^n = X^n \cap A$. This provides the filtration

$$X^0 \cap A \subset X^1 \cap A \subset \cdots \subset X^n \cap A \subset \cdots$$

which is a regular cell complex for $A$.

This describes how we can incrementally construct a spatial representation by adding vertices, then lines, etc., with each stage, every skeleton $X^n$ of $X$ being a subcomplex of $X$. This global construction does not provide a good model for regions which provides a more top down description of how space should be structured. Once the regions have been defined we can then use the skeleton to provide a basis for an implementation and perhaps use the following:

**Definition B.18** ([168, page 72]) Let $c$ be an $m$-cell. Then & is the union of a finite number of $(m-1)$-cells, each the intersection of an $(m-1)$-plane with &. These cells are uniquely determined by $c$.

This is similar to the hyperplane approach described by Günther [92].

The concept of $n$-skeleton helps reinforce the idea of the cellular topology structuring space, in terms of the 1-cell lines providing the skeletal structure of the 2D areas. This makes the structure of space appear planar, and to this planar structure we want to add the potential for hierarchical ordering. To do this we use subcomplexes.

2.2 The Subcomplex

**Definition B.19** ([168, page 21] and [76, pages 33–35]) Let $X$ be a regular cell complex, and let $\Omega$ be a set of open cells of $X$. A regular cell complex $A$ is a subcomplex of $X$, if

1. $A$ is a subspace of $X$ that equal a union of all cells contained in $\Omega$, and
2. for each open cell $e_\alpha$ of $\Omega$ contained in $A$, its closure, $\overline{e_\alpha}$, is also contained in $A$.

When true, $A$ is a closed set in $X$, and a regular cell complex in its own right.

**Definition B.20** ([76, page 35]) Arbitrary unions and intersections of subcomplexes of a regular cell complex $X$ are subcomplexes of $X$. 
From this we can say that a region is both a subcomplex and a regular cell complex, and supported by the following definition we can represent a cell complex with a triangular tessellation.

**Definition B.21 ([168, page 216])** A regular cell complex $X$ can always be triangulated so each closed cell of $X$ is the polytope of a subcomplex.

A *polytope* is a space $|k|$ that is the union of the simplices of $k$ (see Munkres [168, page 8] for details). As described in definition B.10 a simplex can be represented by a cell and we just retain the term simplex to maintain consistency with the definition given in the referenced texts.

### 2.3 The Subdivision

We use subdivision to describe how the database primitives in section 3 affect the structure of space made by the leaf regions that tile the ground-plane. Note that the leaf regions are themselves represented by a cellular implementation layer.

**Definition B.22 ([166, page 70])** A *subdivision* $K'$ of $K$ is a complex such that $|K'| = |K|$ and each simplex of $K'$ is contained in a simplex of $K$.

**Definition B.23 ([76, page 66])** A regular cell complex $X'$ is a subdivision of the regular cell complex $X$ if $X$ and $X'$ coincide as spaces and every cell of $X'$ is contained in a cell of $X$. Also:

- the identity map is cellular as a map $X \to X'$
- if $X$ is finite, locally finite or countable the same holds true for $X$
- $\dim X' = \dim X$.

**Definition B.24 ([166, page 75])** Any cell complex $K$ has a simplicial subdivision.

For more details about simplicial subdivision see Munkres [168, pages 83–88].

### 2.4 Tessellation Techniques

In an irregular tessellation we may have a set of required edges between vertices that describe the extent of environmental features (e.g., walls and typical paths). This set of required edges makes some tessellation techniques such as Delaunay triangulation (Sloan [216], Guibas and Stolfi [91]) inappropriate, because it does not allow special edges to be retained. Although the basic Delaunay triangulation is inappropriate, Sapidis and Perucchio [208] describe an extended Delaunay triangulation algorithm that could be used, since it is able to triangulate the space inside a specified polyhedron.

An alternative to Delaunay triangulation is its dual, called variously a Voronoi diagram or Dirichlet or Thiessen tessellation. A Voronoi diagram produces a tessellation of convex polygons, and can be performed so that it retains the special edges. This tessellation is
likely generate a large number of small polygons that require further tessellation to produce a triangulation. The benefit of this approach is the generation of the medial axis for the special edges providing their retract. We discussed retracts in chapter 2 section 2.2.3 on motion planning. See Latombe [135, pages 169–174] and Fortune [74] for further details, and Hobby [101] for an example tessellation. The properties of the Voronoi diagram can be described in relation to the problem called loci of proximity because its solution results in a data structure that represents the nearest neighbour of each point (Preparata and Shamos [189, pages 204–225]).

Problem B.25 Given a set of $n$ points in a plane called sites, for each point $p_i \in \text{sites}$ what is the locus of points $(x, y)$ in the plane that are closer to $p_i$ than to any of the other $n - 1$ sites.

This collection of loci, structures space into polygonal cells (2-cells) with a 1-skeleton (see definition B.17) being formed by the loci boundary. The Delaunay triangulation results from joining two points $p_i, p_j \in \text{sites}$ if they share a boundary edge (i.e., 1-cell in the 1-skeleton). The Voronoi diagram is the 1-skeleton provided by the boundary edges.

There are other triangulations that have different constraints and which are more appropriate for implementing cells in part because they provide faster linear time triangulation algorithms. For example, Kong et al. [125] describe an algorithm that is linear for simple polygons that have few concave vertices. For more details see El Gindy and Toussaint [65] who discuss how varieties of polyhedral shapes affect the triangulation problem. Various other algorithms are given by Preparata and Shamos [189], and Mehlhorn [157].

As an alternative to forming triangulations, we might only need to form a combination of convex polyhedra (which could then act as a preprocessing stage to linear time triangulation). There are two approaches we can take, the first is to add new points, called Steiner points (which are vertices inside a polygon $P$ but not on the boundary of $P$) as described by Chazelle and Dobkin [37]. The alternative approach is not to add new points and algorithms for this are given by Green [90], Tor and Middleditch [233]. Keil and Sack [118] provide a more detailed comparison, but basically the addition of Steiner points allows an optimal decomposition to be performed.

Once we have completed a tessellation we might find that some of the cells are too large, one option is to use a Barycentric subdivisions (Alexandroff [4]) which operates on convex shapes to reduce the granularity of the component parts by triangulation.

These tessellation techniques provide an insight into the range of additional functionality that could be added were we to change the implementation of cells. Under the current requirements of SPATIAL-LAYOUT much of this functionality would be redundant. However, the use of retracts (via the Voronoi Diagram) is appealing and may prove useful during a knowledge acquisition phase as a framework for modelling a “roadmap” of typical behaviour.
3 DATABASE PRIMITIVES

Here we provide formal definition of the **leaf-split**, **leaf-merge** and **composite-union** operators.

### 3.1 LEAF SPLIT

Figure B.5 illustrates how the **leaf-split** operator subdivides a leaf region. By adding a new leaf region, $A$, we subdivide the leaf region $L$, into the required new leaf region, $A$ and the remainder $B$, such that $A \cup B = L$. At the leaf-region level this changes the structure of space. At the cell level this only alters the structure of the underlying space if a new 1-cell is needed that is not supported by the cell structure.

**Definition B.26** Let $C$ be a regular cell complex, and $X$ a subcomplex of $C$. We can **leaf-split** $X$ by subdividing it into two subcomplexes $A$ and $X - A$, if $A$ is a subcomplex of $X$. We can denote $X - A$ by $B$ such that $X = A \cup B$.

Once $A$ and $B$ have been created we remove $X$ as it is no longer a leaf region since $A$ and $B$ now tessellate its space and leaf regions are not allowed to overlap.

### 3.2 LEAF MERGE

The inverse of **leaf-split** is called **leaf-merge**. At the leaf-region level this alters the structure of space, because the cells that supported the inter leaf boundary are lost. However at the cell-level, no change need take place to the cells that structure space. If much merging were performed then fragmentation may be an issue that would need to be addressed in the implementation.

**Definition B.27** Let $C$ be a regular cell complex, and $A$ and $B$ be two neighbouring subcomplexes of $C$. We can **leaf-merge** two leaf regions $A$ and $B$ if they share a face $x$, such that $\text{Face}(x, A) \land \text{Face}(x, B)$ is true. In this case $A$ and $B$ are subcomplexes of $C$ and $C = A \cup B$.

Once $C$ is created we remove $A$ and $B$ because leaf regions are not allowed to overlap.

### 3.3 COMPOSITE UNION

To define **composite-union** we will use the following terms introduced in chapter 3 section 2.1: $LR$ is the set of leaf regions, $CR$ is the set of composite regions, $R$ is $LR \cup$
CR, and AS is the set of adjacency sets. The function composite-union is of the type composite-union :: R → R → CR. This describes how a composite region can be composed from any combination of two leaf or composite regions.

**Definition B.28** Each cr ∈ CR has a subdivision S where (∀lr ∈ S,(lr ∈ LR)). We call S the leaf-description of cr and call the function that obtains the leaf-description LeafDesc.

In addition to the function LeafDesc we also use the infix operator ∩leaf which has the following type definition (∩leaf) :: LR → LR → {AS}. We use ∩leaf to determine if the intersection of two leaf regions is either empty or a set of adjacency sets. If a set of adjacency sets is returned they describe the continuous boundary between the two regions.

There are two definitions of composite-union which we call “strong” and “weak”. The strong one is more strict, complying with definition B.14 the weaker one does not.

**Definition B.29** A strong composite-union C exists for complex A and B iff

LeafDesc(A) ∩ LeafDesc(B) is a cell complex C and that

(∀a ∈ LeafDesc(A), ∀b ∈ LeafDesc(B), (|a ∩leaf b| = 1) ∨ (|a ∩leaf b| = 0))

The result of the intersection of each of the leaf regions from A and B is either empty or an adjacency set that contains the (n-1)-cells (and lower dimensional cells that connect these (n-1)-cells) that are the face of an n-cell in A and an n-cell in B. This provides the (n-1)-skeleton between the two leaf regions such that it complies with the adjacency set definition.

**Definition B.30** A weak composite-union C exists for complex A and B iff

((LeafDesc(A)^n ∩ LeafDesc(B)^n) ≠ ∅) ∧ ((LeafDesc(A)^(n-1) ∩ LeafDesc(B)^(n-1)) ≠ ∅).

The intersection only takes place along the (n-1)-cell boundary between A and B should they have this boundary. This does not ensure that A and B only intersect in one continuous adjacency set. Removing this constraint is necessary for representing the space covered by some spatial semantics. However, it does not affect the structure of space, since spatial reasoning can still use leaf regions.

In section 1.4 we began an example to describe the definition of adjacency sets and here we extend the example in terms of leaf and composite regions. In definition B.14 the third clause is true for all leaf regions in the ground-plane. It does not hold for composite regions since these are not adjacency sets, instead they are sets of adjacency set symbols. In the case of set operations applied between ground-plane regions and vehicle regions, the time dimension ensures that they are distinct, although address based set operations are possible, since the regions share a common coordinate system and are not based on just the topological cellular relationships.

In figure B.2 if A, B, C, and D described a region, then the region is defined by the set of cells: \{A, B, C, D, \{A, B\}, \{A, C\}, \{B, D\}, \{C, D\}, \{A, B, C, D\}\}. This gives an open 2D area of space because, from the supplied symbolic information, we are unable to name the external boundary.
4 Example spatial layout file

This shows an incomplete data file, which is included to give a flavour of the initial data format. The data file contains three keywords:

**sort-order** defines the ordinal order of types from space down to leaves.

**root** defines the top most entity from which everything is decomposed, in some sense the root is the universal set.

**decomposition** defines how an instance of a type can be decomposed. In this example there is only one instance of each type.

The data file is as follows:

```
(sort-order space usertype subdivisions behaviours names leaves)

(root world)

(decomposition space
  (world (bremer-stern-roundabout other)))

(decomposition usertype
  (bremer-stern-roundabout (pavement reservation tram-route
    road-surface cycle-path)))

(decomposition subdivisions
  (road-surface (lane roundabout turning))
  (cycle-path (cycle-lane cycle-roundabout))
  (tram-route (tram-line))
  (reservation (centre-of-roundabout concrete-island)))

(decomposition behaviours
  (lane (fast-lane slow-lane lane-giveway lane-giveawayset
    lane-stopping))
  (roundabout (sector giveway-to-tram rbt-giveawayset)
    (turning (turn-right turn-left)))

(decomposition names
  (fast-lane (north-fast-laneR1)))

(decomposition leaves
  (north-fast-laneR1 (R151 R152)))
```

The data file is intended to create a type lattice (for details about the type lattice see Cohn [42] and Cohn [43]). The type lattice is used to define the leaf regions connectivity and the hierarchy of composite regions.
5 Cell parser

Here we will discuss how the cell-types outlined in chapter 3 section 3 can be used to provide a better model of space than that provided by just using a grid of cells. To do this, let us first look at the original implementation, and then compare this with the new one.

In the original grid of cells implementation, for two regions to meet at a boundary, the boundary was (conceptually) represented by the common edge of the adjacent cells along the boundary. That is, we are using the 1D edge cells of the 2D square cells and, since we can only implement the 2D square cells, reasoning about the boundary becomes a little bit more difficult. However, a 1D cell can be completely defined by its two adjacent 2D cells. This model of a region's boundary works fine for event detection because, topologically, the important change is when an object enters or leaves a region. This means that we do not reference the boundary, instead we reference the space filled by the region.

Cell-types were used in an attempt to make the boundary more explicit. The basic cell-types describe the different cells that "compose" a regular square cell, because every (n-1)-cell must be a face of at least two n-cells. This was the condition that defined the boundary in the first regular square cell implementation. To make a boundary explicit involves adding more points of space to "close" the edges of the boundary. Another way of viewing this is to consider the boundary running along the centre of the square cells, cutting the cells it goes through either in half or into a $\frac{1}{4}$ and a $\frac{3}{4}$ pair. It is these divided cells or boundary edges that, in the implementation are sharing the same cell (the implementation's way of adding more space) while maintaining a separation between the two neighbouring regions. This means that the edge cells from two neighbouring regions do not intersect. This has the nice property of making the structure of space explicit. Unfortunately, the extra effort involved in producing a model of space using cell-types does not really improve the results of event detection.

The two models are both based on regular square cells and as shown in figure B.4 correspond to two different (though equivalent) ways of describing space. These are the closed-edge and open-edge models of space. The first model is open, since we have ignored the space that makes up the boundary, so that the cells either side of the boundary are adjacent (in the implementation). The second model is closed, since we have added extra space to define the boundary.

Neither of these approaches solve the major issue of storage, in fact the second method increases it slightly. On the positive side, the use of cell-types gives a better model of how cellular topology structures space. To form the model of space using cell-types involves tracing around the boundary between regions completing the space on either side of the boundary. To do this, we have used the transition network in figure B.6, which specifies the legal order for building up the closed boundaries simultaneously. The transition network is ambiguous and to make it deterministic involved deciding between the allowed transitions by using lookahead. This lookahead uses a finite state machine for matching the next cell to be drawn, as well as checking to see what type of cell already exists in the cell (address)
to be written to. The finite state machine has states for the eight compass directions, a beginning, and terminal state, with the arcs between these states defining what is written and where. All this has been hardwired into a table driven CELL-PARSER. Although the CELL-PARSER has been useful from the standpoint of showing how the theory works, it has provided little practical benefit.

The CELL-PARSER was used to obtain the cell description shown in figure 3.6. The cell types described in figure 3.7 are composed to provide the different nodes in figure B.6. For example, node 1 is composed from cell-types j and e. This composition produces a closed boundary because as described in figure 3.8 the two cell-types do not intersect.
6 Tolerance Space

In Hayes' [95] naive physics manifesto there is a subsection headed "Qualities, Quantities and Measurements" describing how some everyday things have properties, called "qualities", that are more intrinsic than others. Qualities are similar to the spatial semantics introduced in chapter 3, except that spatial semantics concern the behaviour that takes place in a space and qualities concern the physical properties of a space (e.g., weight, colour, size, volume, smell, etc). To reason about qualities Hayes uses a metric based on similarity that denotes two qualities as being indistinguishable when they are very similar. Here indistinguishability is a tolerance relation and we call a space defined by such a metric a "tolerance space".

Definition B.31 A tolerance space and is a space that has a collection of cells with a symmetric and reflexive (and intransitive) tolerance relation defined on it.

Tolerance provides a natural notion of distance between qualities, describing the smallest number of "steps" by which one quality can be transformed into another, with each step being invisible under the tolerance. This is because most quality spaces are dense (i.e., for any two distinct values there is a third). For example, consider the colours Red and Blue, and how we could transform from one to the other in such small steps that the difference between the steps was not distinguishable. The tolerance relation was initially developed by Poincaré [187], who used it to provide a way of expressing "if A is near B and B is near C, then A is not necessarily near C". For example, using colours again, consider A representing Red, C representing Blue, and B representing Purple.

This subject has been investigated by Kaufman [117], Davis [50, 51], Poston [188], Dobson [58], Roberts [201], Zeeman [262]. Poincaré used the term "physical continuum", Zeeman the term "tolerance space", and Poston the term "fuzzy space". They describe the nature of our perception of space, and have adapted it to tools current in topology. Unlike the Euclidean plane or pseudo-Riemannian manifolds fuzzy spaces occur as part of direct experience. The philosophy is that limits are physically meaningless and mathematically unnecessary, and the ideal is to displace Euclidean geometry in the physically small.

The cellular topology uses the concept of error neighbourhoods (see Fleck [68, page 51]) when comparing the values at two cells. The two values are indistinguishable if their error neighbourhoods overlap. Fleck uses this to define a digitise function for mapping a continuous function to a discrete one.

We can apply tolerance space to the relationship between the camera's view and the ground-plane, and the description of objects.

6.1 Camera View

We can add tolerance neighbourhoods to positions further from the camera to provide a spatial equivalent to its visual fuzzyness with the size of the neighbourhoods being proportional to distance from the camera. This would enable us to represent in the ground-plane the effect of the camera position. In the ground-plane regions near the camera would
be clearly defined while regions further away would have less clear boundary edges. Regions even further away may not have vertices that are distinguishable from each other or from those of neighbouring regions. The general idea is illustrated in figure B.7.

The 2D ground-plane could be enhanced to include a representation of the error neighbourhood due to occlusion by scene features. For example, in figure B.8 the hump in the central reservation conceals the road surface behind it. To represent this would require a better mapping between the image-plane and ground-plane, or an additional database of 3D scene information.

The camera image also bounds the perceiver’s field-of-view. Figure B.7 shows areas where objects enter and exit the field of view. We could add neighbourhoods to these edges of the image-plane and map them onto the ground-plane to indicate the increase in errors associated with these parts of the scene where objects suddenly appear or disappear.

6.2 Objects

Applying the concept of neighbourhoods to the posebox descriptions of objects is likely to obtain something similar to the proximity description given in chapter 4 section 6.2.1. Using neighbourhoods at larger scales (or extents) could be used to identify groups, and at smaller scales to describe when two objects are indistinguishable (maybe due to collision).

We could combine this posebox error neighbourhood with that used for the ground-plane to describe how object position data increases in noise with its distance from the camera. Objects close to the camera are resolved accurately while objects much further away may only have a coarse description of position because the vertices of the posebox are indistinguishable from one another. Combining these properties of how space is perceived would result in a more realistic system, but at the cost of more complex implementation and runtime overhead.
APPENDIX C

SPATIO-TEMPORAL MECHANISMS

In this appendix we describe the mechanisms that support the spatial reasoning described in chapter 4. There are three sections that deal with time, spatio-temporal representation and spatial arrangements. The work described here builds upon the spatial representation described in appendix B.

Temporal reasoning is not central to the approach taken in this dissertation but it is an important component. For consistency, we adapt the cellular topology to the 1D time-line to provide the “cellwise-time” language which we have used in chapters 4 and 5. We begin the description of spatio-temporal mechanisms by describing the syntax and semantics of cellwise-time providing two equivalent notations.

Spatio-temporal representation concerns the motion of the dynamic objects in the scene. We describe some approaches for modelling such paths and also for how their spatio-temporal development can be predicted. Having developed cellwise-time we explore the opportunity for using cellular topology as a unifying spatio-temporal representation.

Spatial arrangements covers some implementation issues for operators described in chapter 4.
Figure C.1: State and action models of situations using cellular topology. The shaded cells can be referred to using a progressive form.

1 TIME

In this section we provide some background material on temporal representation and connect the representation of time to the representation of space that we have already described to provide the cellwise-time model. This temporal representation is used in HIVIS-MONITOR to assert the temporal information produced at runtime in a temporal database called the Temporal History Builder which we describe in chapter 4 section 3.3.2.

1.1 REPRESENTATION ISSUES

Time is different from space in that it flows in one direction, i.e., there is an order defined on it. This means that we can distinguish the past from the future and that the order of events cannot be permuted once they have happened. However, it is appealing to use the same form of basic representation for both, allowing us to merge the two into a uniform representation of space and time. In this section of the appendix we will discuss how the cellular topology, already introduced, can be applied to temporal representation and reasoning. The purpose of this is to develop a language that we use in this discussion of the temporal properties of the surveillance problem.

Fleck develops a natural language semantics [68, 224-276] that uses a four-way classification of situation, used for some time in linguistics and originally due to Vendler and which van Benthem also calls the “Aristotle-Kenny-Vendler” verb classification (see [244, pages 133 and 196]). The situations are divided into: “activities”, like eating, driving a car; “accomplishments”, like travel to work, after an hour’s debugging the program compiled; “states”, like own a house, be a student; “state-changes”\(^2\), like get to work, find a book. These situations are described pictorially in figure C.1, which also shows another situation called “episode” to denote accomplishments that have a definite beginning and ending. The term “actions” is used as a cover term for activities, accomplishments, state-changes and episodes. And according to our ontology introduced in chapter 2 section 3.2.1, “events” are a semantically meaningful subset of the set of all state-changes. Similar classifications

\(^2\)Also called “achievements”.
have been used by Ryle [206], Taylor [229], and Allen [5]. Here time is modelled using a
cell complex whose underlying space is the real number line \( \mathbb{R} \). As shown in figure C.1 we
can build the more complex situations from the three basic forms, states, activities and
state-changes. A state describes the properties of the world at a single cell in time, and
is represented by a cell which holds some set of measured properties. An activity denotes
some process where some continuous change is performed, and that is represented by a
connected interval of time that contains at least two cells. A state-change describes an
abrupt change in properties, with an associated boundary in time, between two cells.

From these we can compose a accomplishment, as a sequential ordering of an activity
and a state change, which denotes the sequence of a continuous change followed by a final
result. As shown in figure C.1 an accomplishment might have a state change that starts
the activity, and we make these situations distinct by adding the situation “episode”. An
episode is described by composing a state-change denoting the start of the episode, an
activity denoting the actions that take place during the episode, and an final state-change
denoting the end of the episode (see figure 5.19). This allows us to refer to the process
during an episode as an activity, the properties that initiated and concluded the event as
well as the accomplishment of completing the event. An example of an accomplishment is
“riding my bike to work” which begins with me getting on the bike at home, performing
the continuous activity of riding the bike by which I get progressively closer to the place
where I work, and a final state-change of arriving at work. To determine that a state-
change has really happened we need to check that the state following the state-change is
different from the state that preceded it.

Fleck [68] places this in a linguistic context, which we will not discuss here because it
is outside the scope of this thesis, however it does provide one route for incorporation of
a natural language interface. The models of situations provided by the cellular topology
avoid certain problems that previous researchers have encountered. We will only briefly
discuss these problems here, referring the interested reader to Fleck [68, pages 256–267]
for further details.

The distinction between states and actions as shown in figure C.1 is important. State-
changes, accomplishments and episodes are easily distinguished from states because they
all contain at least one distinctive boundary. This is not the case when we consider
activities, which closely resemble states in terms of semantics. There are three approaches
to distinguishing states from actions, called cellwise, pointwise and interval-axiom. We
will use the cellwise approach which is:

**Definition C.1 ([68, page 257]) Cellwise proposal:**

A state is a description that can be verified for individual cells.

An action is a description that can only be verified for intervals containing at least two
distinguishing cells.

The pointwise approach used by Taylor [229] is similar to the cellwise one.
Definition C.2 ([68, page 257]) Pointwise proposal:

A state is a description that is true of individual points.

An action is a description that is only true of intervals.

In the pointwise proposal it is necessary to stipulate that an action which holds over an interval is larger than a single point, i.e., an interval needs to include two end points. because this proposal allows zero duration intervals. The cellwise approach does not have this problem because intervals are represented by sets of cells. Both the cellwise and pointwise proposals represent the common notion that to verify an action has occurred requires at least two distinct measurements, and which is the case with the stream of compact encodings provided by the intermediate-vision component. The cellwise proposal also better represents how measurements can only “pin down” the state of the world over some finite width of time and not sample the world at infinitely small points.

In AI the actions and state are distinguished from each other by axioms that describe the relationships between truth and intervals. For example Allen [5] proposes:

Definition C.3 ([68, page 258]) Interval-axiom proposal for distinguishing states from actions:

Both states and actions are descriptions of intervals.

If a state is true of an interval, it is true of all of its subintervals.

If an action is true of an interval it may not be true of its subintervals.

Shoham [214] takes a similar approach offering a finer classification, pointing out that in the interval-axiom proposal although the inclusion of subintervals seems appropriate for describing facts, it is less clear whether an assertion that holds over an interval holds over the parts of that interval when we consider activities, because if we look at them more closely they may be subdivided, for example, the activity eating may include snippets of conversation, time between courses, etc. The interval-axiom, as it stands, is not able to distinguish states and activities from accomplishments and state-changes, as was possible in the cellwise case, so we need to extend the interval-axiom with:

Definition C.4 ([68, page 259]) Interval-axiom proposal for distinguishing states and activities from accomplishments and state-changes:

If a state or activity is true of two intervals it is true of their union.

If an accomplishment or state change is true of two intervals, it is not necessarily true of their union.

This extension to the interval-axiom proposal is not perfect, having two faults. First, although the proposal holds for states, it is not clear that it is also true for all activities, which by their nature are not static, involving continuous change, for example, when drinking coffee the mug is continuously part of the routine of being picked-up, drunk-from and put-down, during which its contents get depleted. Secondly, the distinguishing property for accomplishments and state changes may seem too strong, since it depends upon how the two accomplishments or state changes interact. Certain accomplishments
are composed of smaller ones, for example, when a young child is at the early stages of coordinating walking, the accomplishment of putting one foot in front of another is repeatedly composed to accomplish walking across the room.

The cellular topology provides the choice between two boundary representation, which although better than ad hoc rules for assigning the boundary point, do not provide a perfect solution. The boundary forms are:

- Closed at left and right. [ ] “closed-boundary model”
- Closed left, open at right. [ ] “beginning time”
- Open at left, closed at right. [ ] “ending time”
- Open at left and right. ( ) “open-boundary model”

In fact Rescher and Urquhart [197, page 168] rule out these two symmetric boundary models, because it can not be that all propositions are true in such intervals, preferring instead to operate with beginning time, and note that Aristotle argued that all changes, process and motion can have a completion or ending but no start or commencement, i.e., the ending time model. Using "switching on a light bulb" as an example, in the open-boundary case we have a point between the bulb being off and on, where the bulb is neither on or off, and in the closed boundary case, the bulb is both on and off at the same time (which is a contradiction). Williams [255] describes how under temporal magnification the symmetrical open-boundary model can be thought of representing the continuous change in the light bulb’s filament as the current increases. We do not require the precision provided by temporal magnification in the surveillance problem, but we still need to model changes. Shoham [214, page 34] says that deciding whether an interval contains its end points is meaningless, viewing an interval as allowing the set of points that lie between the end points of an interval not to be mutually comparable. This is a useful position to take in the surveillance problem.

When we consider modelling sharp state-changes, the cellwise model allows this with the representation shown in figure C.1 consisting of two cells, one on each side of the state-change. This method of modelling state-changes is not available if time is modelled as \( \mathbb{R} \) without cell structure, because there is no natural definition of a finite "minimal" size for an interval. There could always be another interval between the faces of the boundary formed by the state-changes. The cell size provides a limit is called the "granularity" to our representation (see Hobbs [100]). The cells need not all be the same size, but their size should all be distributed about some fiducial non-negligible (though short) value.

Shoham [214, pages 32-37] discusses the initial choices open to us when we are considering what features we want to be present in our temporal representation. There are four areas: (1) the selection of how to interpret assertions, the choice is usually between using a time-point or a time-interval to represent \( t \) (see van Benthem [244] for a thorough introduction), we will not allow assertions to be interpreted over time-points, instead we will allow interpretations over activities, which must include at least two cells, and states; (2) we then decide what are the primitive temporal objects, since we have chosen to make
assertions over activities and state, we can now select between using time-points or time-intervals as our primitives, for example, if we were going to represent the endpoints of a time-interval, i.e., \( t = (t_1, t_2) \), we will take cells as our primitives and define activities and states in terms of these; (3) we have already discussed the relation between the truth of an assertion over an activity and its truth over parts of that interval; and (4) the selection of precedence (time is usually considered to be linear and acyclic), and between discrete vs dense, complete vs incomplete, and bounded vs unbounded, we have described above the underlying time-line is dense and complete and bounded (each point has an earlier and later point), although the addition of the cellular structure provides discrete properties that are useful to our surveillance problem where a stream of discrete frame-updates are provided, providing an initial structure to our time-line.

1.2 Cellwise-time language

We want to associate an atemporal property with an activity or state and call this a steady-state and also call the value changes between temporally adjacent cells of the same property a state-change. A straightforward way of doing the first requirement is described by Shoham [214, page 41], where he simply forms an interval-property pair. Once we have defined out temporal language using this primitive formula we will use it to define the state-change model.

1.2.1 The propositional and first-order notation

Each primitive formula is denoted by a pair \((i, p)\), where \(i\) is an interval symbol and \(p\) is a primitive propositional symbol. Intervals are replaced by states and activities in the cellwise-time language, the interval symbol \(i\) is really a pair \((t_1, t_2)\), where the \(t_1\)'s are time-cell symbols. Following Shoham, we will replace \((t_1, t_2, p)\) by \(\text{TRUE}(t_1, t_2, p)\).

The syntax of this propositional language is given as follows: \(P\) is a set of primitive propositions, \(TC\) is a set of time-cell symbols, \(TV\) is a set of temporal variables, \(U\) is \(TC\cup TV\), and with \(\leq, =\) being our binary relation symbols. The set of well-formed-formulas (wff) is defined inductively as follows:

- If \(u_1, u_2 \in U\) then \(u_1 = u_2\) and \(u_1 \leq u_2\) are wff.
- If \(u_1, u_2 \in U\) and \(p \in P\) then \(\text{TRUE}(u_1, u_2, p)\) is a wff.

This has described the syntax of our simple cellwise-time propositional language, we will now describe its semantics.

The semantics describe an interpretation as a tuple \(\langle TW, \leq, M \rangle\) where \(TW\) is a nonempty universe of time-cells, \(\leq\) is a binary relation on \(TW\), \(M = (M_1, M_2)\) is a meaning function, with \(M_1 : TC \rightarrow TW\) and \(M_2 : P \rightarrow 2^{TW \times TW}\). We require that \((w_1, w_2) \in M_2(p)\) iff \((w_2, w_1) \in M_2(p)\) so that a pair of cells denotes a single interval.

A variable assignment is a function \(VA : TV \rightarrow TW\). An interpretation \(S = \langle TW, \leq, (M_1, M_2) \rangle\) satisfies a wff \(\varphi\) under the variable assignment \(VA\) (written \(S \models \varphi[VA]\))
given the following inductively defined conditions (in the following, for any \( u \in U \), \( \text{VAL}(u) \) is defined to be \( M\!L(u) \) if \( u \in T\!C \) and \( \text{VA}(u) \) if \( u \in T\!V \)):

- \( S \models (u_1 = u_2)[\text{VA}] \) iff \( \text{VAL}(u_1) = \text{VAL}(u_2) \)
- \( S \models (u_1 \leq u_2)[\text{VA}] \) iff \( \text{VAL}(u_1) \leq \text{VAL}(u_2) \)
- \( S \models \text{TRUE}(u_1, u_2, p)[\text{VA}] \) iff \( \langle \text{VAL}(u_1), \text{VAL}(u_2) \rangle \in M\!E(p) \)

An interpretation \( S \) is a model for a wff \( \varphi \) (written \( S \models \varphi \)) if \( S \models \varphi[\text{VA}] \) for all variable assignments \( \text{VA} \). A sentence is a wff containing no free variables.

This completes the definition of our propositional cellwise-time language, using the same definition framework as Shoham, and we can extend this simple language to a first order form using an approach similar to that described by Shoham [214, pages 43-46]. In this thesis most of the examples use this first order temporal logic, however, we do not describe the derivation here because this task is admirably described by Shoham who explains the subtleties involved in performing this extension.

### 1.2.2 Incorporating Observations

In the surveillance problem we are given a stream of discrete observations (with each observation being a frame of compact encodings). These observations provide the major discretisation to the time-axis. Although we have defined a state-change to be a signal-change (for example, a change in a feature of a stimulus array; and for details see chapter 2 section 3.2.1), and that any feature change is a necessary and sufficient condition for an event to be perceived, these signal-changes do not need to coincide with the observation updates. The cellwise-time language can describe how a state or activity holds over discrete observation, but it is not able to model the position of state-changes. We will now try to address this problem.

In the cellwise-time language we can represent steady-states, and one approach of expressing a state-change is to use a relation symbol, such as, CHANGE to denote a state-change. The trouble with this is that it is asserted to occur over one or more cells when in fact it occurs between two cells (i.e., it refers to a boundary). One solution to this is to alter our interpretation of intervals when we deal with state-changes, because, as already described, a state-change is a boundary between two cells and the reference to the two cells just denotes the occurrence of the boundary between them. For example, a house remains structurally the same after it has been painted as different colour, only the property colour has changed. Say we see the house during an interval when we walk past it on Monday \((t_1, t_2)\), and on a subsequent day when we walk past again, \((t_3, t_4)\), its been painted a different colour, which must have occurred some time during the interval \((t_2, t_3)\).

\[
\text{TRUE}(t_1, t_2, \text{COLOUR(HOUSE17, RED)}) \land \text{TRUE}(t_3, t_4, \text{COLOUR(HOUSE17, BLUE)})
\]

\[
\Rightarrow \text{TRUE}(t_2, t_3, \text{CHANGE(\text{COLOUR(HOUSE17, RED), COLOUR(HOUSE17, BLUE)})})
\]

Altering the interpretation of our cellwise-time language should be avoided, and in the discussion that follows we will show that it is unnecessary.
In addition to expressing this problem, this example also introduced the description of property values that are associated with an object. In our example we are dealing with the object property **COLOUR** which has the values **RED** and **BLUE**. In the **surveillance problem** we encounter a number of object properties so it is worth generalising this representation to some object property \( P \) that has a finite set of \( n \) qualitative values \( \{v_1, \ldots, v_n\} \), for which there are \( n^2 \) different state-changes. Although we can say which state-change has occurred, we are having difficulty expressing when it occurred. To solve this let us reassess the problem.

In the **surveillance problem**, we get our frame updates a bit quicker than in our house painting example, but they still have a noticeable interval between updates. The camera operates at 25Hz and we get a compact encoding of what occurs in every tenth or twelfth frame, giving us one frame every 400 ms or 500 ms (which is 2.5 fps or 2 fps). As in the house example, changes often appear to take place between frames. Newton [175] describes a number of experiments concerned with how people attribute points of change to continuous everyday experience. He differentiates between “break points” where one meaningful action ends and a different one begins\(^2\) and “nonbreak points” which make up the continuous process of some action. It seems likely that there are, in general, more nonbreak points than break points. However, as Newton’s results show, it is more difficult to understand a scenario from nonbreak points than break points (the break point description picks out the critical summarisation points in behaviour), and because of this it is useful to try and extract the break points (i.e., the semantically important state-changes we are calling events).

1.2.3 The Cellular-Topology Notation

To investigate how to apply this to our state-change definition, let us first introduce an alternative notation based upon figure C.1 which better describes the cellular structure of the time-axis. This notation is useful, because in some examples and definitions it is easier to express the the time axis in terms of cells and boundaries, to display the property values that hold at a given cell in a state or action, and to make explicit the boundary of the state-change. For example, if a property, \( P \), has a range of values \( \{a, b, c\} \) we can have examples like:

\[
\begin{align*}
(\langle c \rangle) \langle () \rangle \\
(\langle a \rangle, \ldots \langle () \rangle \\
(\langle b \rangle, \langle b \rangle, \ldots \langle () \rangle \\
(\langle a \rangle, \ldots \langle a \rangle, \langle b \rangle, \ldots \langle b \rangle, \ldots \langle () \rangle
\end{align*}
\]

These examples illustrate how the values of a property, \( P \), vary over time. The round brackets represent a time cell. The symbols inside the time cell are the property values. The dots denote the possibility of more time cells that have the same value. The square brackets delimit a continuous signal and this notation is used to describe a boundary

\(^2\)We called this the “point-of-action-definition” in chapter 2 section 3.2.2.
between two different but temporally adjacent sets of values, and is denoted by $\mathcal{I}$. The empty round brackets, ( ) or ( ? ), denote no value for this property and are used to express the continuous nature of time. The first example shows a state with the value “$c$”, the second a state or activity with the value “$a$”, the third an activity with the value “$b$”, and the fourth shows the change from “$a$” to “$a$ $b$” to “$b$”. The property in this example can have two values at the same time, for example, the values might represent the names of people occupying a room.

To be more explicit about values not being present in a time-cell we use the notation $\neg$ to mean “not present”. For example, $(\neg b) \mathcal{I}(b) \mathcal{I}(\neg b)$ represents a property that has a set of values that start off not including “$b$”, then includes “$b$”, then not again. Let $p_t$ be the set of property values, for property $P$, at time-cell $t$, then

- $\models (\neg b)$ iff ($\forall z \in p_t, x \neq b$)
- $\models \neg(b)$ iff ($\exists z \in p_t, x = b$)

Which provides a rough semantics for this notation. Note that the name of each time-cell $t$ on the time-line is implicit in this notation, and that this notation is assumed to represent a nonempty universe of time-cells ($TW$) that has a $\leq$ binary relation (denoted by the cells being written left to right).

### 1.2.4 Completing the Language

The cellular-topology notation allows us to resolve the interpretation problem raised in section 1.2.2. We can redefine the essence of our house painting example as: $(\text{RED}) \mathcal{I}(\text{BLUE})$. This denotes the change in colour of the observed house. We can rewrite this to include an interval of new cells $(u_1, u_n)$ between the two observations. This allows us to say that the change occurred during the interval that denotes the time between observations. The rewritten form is:

$$(\text{RED}) \mathcal{I}() \ldots () \mathcal{I}(\text{BLUE})$$

$$_{t_2} u_1 \quad u_n \quad v_3$$

Where $(u_1, u_n)$ contains the activity of change where the house was painted BLUE. The addition of more cell structure has provided cells to which we can attach information about the activity that produced the observed change (for example painting). It also supports a qualitative model for more precisely describing when a particular change could have occurred via extrapolation. The new “extrapolation cells” are different from the “observation cells” because they do not hold any observed properties. To make use of this new cell structure we require three extensions to our cell-wise-time language.

The additional syntax is:

- $OU$, called “observation updates”, is a set of time-cell symbols such that each contains observed properties and $OU \subseteq TC$;
- $\prec_{PC}$ is a binary relation symbol for previous cell in TC;
- $\prec_{PC}$ is a binary relation symbol for previous cell in $OU$.  


The relationship of OU and TC is illustrated in figure C.2, and their semantics are:

- \[ S \models (u_1 \prec_{PC} u_2)[VA] \text{ iff } \forall \alpha \in TC, \text{ VAL}(u_1) < \text{ VAL}(u_2) \land \text{ VAL}(\alpha) < \text{ VAL}(u_1) \]

- \[ S \models (u_1 \prec_{POC} u_2)[VA] \text{ iff } u_1, u_2 \in \text{ OU} \land \]
  \( (\forall \alpha \in \text{ OU}, \text{ VAL}(u_1) < \text{ VAL}(u_2) \land \text{ VAL}(\alpha) < \text{ VAL}(u_1)) \)

We can now describe the house painting example as:

\[
\text{TRUE}(t_1, t_2, \text{COLOUR(HOUSE17, RED))) \land \text{TRUE}(t_3, t_4, \text{COLOUR(HOUSE17, BLUE)))} \land \\
(\exists u_1, u_2 \in \text{ TC}, t_2 \prec_{PC} u_1 \land u_2 \prec_{PC} t_3 \land u_1 \leq u_2 \land t_2 \prec_{POC} t_3 \\
\Rightarrow \text{TRUE}(u_1, u_2, \text{CHANGE(\text{COLOUR(HOUSE17, RED)}, \text{COLOUR(HOUSE17, BLUE)))})}
\]

Which defines an interval during which the change occurred. This definition does not fully describe state-changes because we cannot refer to them directly only to the two cells on either side of the boundary. However, this extension does not need to describe a state-change as occurring over two consecutive OU time-cells, which is an improvement over our initial version. From this discussion, there are two rules that we apply to our logical propositions. We can describe a state-change as:

**Definition C.5**
\[ \forall t_1, t_2, t_3, t_4, \ t_1 \leq t_2 \land t_2 \prec_{POC} t_3 \land t_3 \leq t_4 \land \\
\text{TRUE}(t_1, t_2, p_1) \land \text{TRUE}(t_3, t_4, p_2) \land \\
(\exists u_1, u_2 \in \text{ TC}, t_2 \prec_{PC} u_1 \land u_2 \prec_{PC} t_3 \land u_1 \leq u_2 \land t_2 \prec_{POC} t_3 \\
\Rightarrow \text{TRUE}(u_1, u_2, \text{CHANGE(p_1, p_2)})}
\]

We can describe value continuity as:

**Definition C.6**
\[ \forall t_1, t_2, t_3, t_4, \ t_1 \leq t_2 \land t_2 \prec_{POC} t_3 \land t_3 \leq t_4 \land \\
\text{TRUE}(t_1, t_2, p_1) \land \text{TRUE}(t_3, t_4, p_1) \\
\Rightarrow \text{TRUE}(t_1, t_4, p_1)
\]

The intention is that these two formula are applied over the set of evolving updates that are asserted. This will enable us to identify continuous sequences of cells that have the same value (e.g., activities) and also to identify state-changes. This fulfills the two objectives of our temporal representation language. However, this does not support the construction of episodes, which requires rules of composition and additional operators to act upon the rules. We will describe one such mechanism after we briefly wrap up some final issues concerning temporal representation.
### Figure C.3: Time intervals described using cellwise-time language.

<table>
<thead>
<tr>
<th>X</th>
<th>current Interval</th>
<th>intervals</th>
<th>cellwise-time description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>(7)(XY)(XY)(X)(X)(X)</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td>(7)(X)(X)(X)(X)</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td>(7)(X)(X)(X)</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>(7)(XY)(XY)</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td>(7)(Y)(Y)</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td>(7)(Y)(Y)(X)(X)</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td>(7)(X)(X)(X)(X)</td>
</tr>
<tr>
<td>L</td>
<td></td>
<td></td>
<td>(7)(X)(X)(X)(X)(X)</td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td>(7)(Y)(Y)(Y)(Y)(Y)</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td>(7)(Y)(Y)(Y)(Y)(Y)</td>
</tr>
<tr>
<td>L</td>
<td></td>
<td></td>
<td>(7)(Y)(Y)(Y)(Y)(Y)</td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td>(7)(Y)(Y)(Y)(Y)(Y)</td>
</tr>
</tbody>
</table>

#### 1.3 INTERVALS

- **B**: interval Y is beginning and during interval X
- **E**: interval Y is during and ending interval X
- **A**: interval Y is beginning and immediately after interval X
- **B**: interval Y is beginning interval X
- **E**: interval Y is ending interval X
- **A**: interval Y is ending immediately before interval X
- **D**: interval Y is during interval X
- **L**: interval Y is some interval later than interval X
- **O**: interval Y overlaps the future of interval X
- **D**: interval Y completely overlaps interval X
- **L**: interval Y is some interval earlier than interval X
- **O**: interval Y overlaps the past of interval X

For details about the duality of points and periods see Tsang [236], van Benthem [244]. There are a number of approaches to representing temporal logic, and we have chosen the one is based on Shoham's approach [214]. Not all researchers use the same ontology for example, in Kamp's description of time [114] events are similar to what we have called episodes and instants are similar to cells.

Tsotsos describes the temporal requirements of the ALVEn project in [237, 239], which are similar to those described here, but with additional concepts. These temporal concepts include: rates of change during an event, and temporal constraints between events. Making
P1 \text{Holds(before(u,e))} \\
\quad \text{if Terminates(e,u)} \\
\quad \text{Terminates(e in-region(r,x))} \\
\quad \text{if Act(e enter-region(r)) and Object(e,x)}

Holds expresses a relationship that holds for a particular time period. Terminates is an application specific rule, saying that the act of exiting a region ends an in-region event.

P2 \text{Holds(after(u,e))} \\
\quad \text{if Initiates(e,u)} \\
\quad \text{Initiates(e in-region(r,x))} \\
\quad \text{if Act(e enter-region(r)) and Object(e,x)}

Initiates is another application specific rule, which states that the act of an object $x$ entering a region starts an in-region event.

P11 \text{Incompatible(in-region(r,x) in-region(r',x'))} \\
\quad \text{if or (not r = r') (x = x')}

Incompatible is the third application specific rule.

Figure C.4: Extension to the event calculus to describe region crossing. In this figure, $x$ is the name of an object represented in 2D space by a centroid.

these extensions explicit in the temporal mechanism is likely to be necessary if more complex temporal reasoning is necessary.

Other approaches investigated include the "event calculus" of Kowalski and Sergot [127]. The key idea here is to deal with local events and time periods. This overcomes the frame problem of situation calculus (McCarthy and Hayes [154]) because instead of using global states, the event calculus deals with local events and time periods. Events are topological, and so map well to Fleck’s cell based interval structure. For example, the following axiom supports this similarity.

**Axiom C.7** Any two periods associated with the same relationship are either identical or they are disjoint.

This axiom provides an equivalent way of saying that the model of time has a Hausdorff topological property (one of the requirements for cellular topology). This similarity is also shown by the simple extension described in figure C.4. The machinery described in the figure is very similar to region event detection described in chapter 4 section 2.1. The application of the rules shows this clearly.

This is appropriate for the special case where events are recorded in the order in which they occur and the database (and the time between $t$ and $t'$ is not too long for the relationship concerned to hold continuously. However it is incorrect in the general case because our definition of the “End” (as well as “Start”) predicate is incomplete. Kowalski and Sergot go on to explain how this can be solved in their section titled “Other cases of the Start and End predicates” [127, page 90], which deals with incomplete definitions of events and how they can be resolved using implicit endpoints.
2 SPACE AND TIME

In this section we present details about the interval tree mechanism which is used to both store temporal histories and speed access to the regions held in SPATIAL-LAYOUT. Following this we describe how the paths of dynamic objects can be modelled.

2.1 INTERVAL TREE

There are two applications of the interval tree in the HIVIS-based systems described in this dissertation that provide: more time efficient access to regions in SPATIAL-LAYOUT; and storage of events and construction of activities (as intervals) in the THB. In chapter 3 section 3.6 we describe how bounds are used to represent the space occupied by a region in terms of intervals on the orthogonal axes. In chapter 4 section 3.3.2 we describe the THB. SPATIAL-LAYOUT uses a static form of interval tree because we do not currently alter the regions held in SPATIAL-LAYOUT at run time. This is not the case with the THB which requires the dynamic form of interval tree, which because it has the more general form we will describe it here.

The dynamic interval tree (see Preparata and Shamos [189, pages 359–363] and Mehlhorn [157, pages 192–199]) is used to provide an implementation of THB. It acts as a database for storing intervals during which some property value is continuous. The power of the interval tree approach is its ability to efficiently return the “names” of all intervals that contain a common time cell by cutting through the mutually independent asserted property values that include this time cell. This functionality plays a key role in identifying the presence of an episode. The interval tree is used both to store state and activity durations and for multiple-object-reasoning described in chapter 4 section 2.3. Figure C.5 shows how the interval based approach fits well with the spatial reasoning.

The interval tree is used in two stages, the first is building up the temporal history, the second is composing the base-data intervals into events and episodes. This composition process in an incremental process and is driven by the data and the scripts that the parser can recognise. This parser stores temporal intervals and performs pattern matching using the overlap, inclusion and meets operators of Allen [5].

\[
\begin{align*}
\text{meets}(A,B) & \quad (A)[[B]) \\
\text{overlap}(A,B) & \quad (A)[[A B]][B) \\
\text{inclusion}(A,B) & \quad (A)[[A B]][A)
\end{align*}
\]

meets is used to describe a state-change of a property. overlap and inclusion are used for composing activities and states into episodes. These operators are used to reason about different values of the same property, and the values of different properties.

In addition to the construction functions, the THB also has functions for deleting and matching data. The matching of episode patterns uses the ability of the interval tree to: (1) report all intervals in the tree that intersect a given interval; (2) report all intervals in the tree that enclose a given interval; and (3) report all intervals in the tree that are enclosed by a given interval.
One of the key features of the time line description is the use of an interval tree which manages the history of each scene object. All object information is inserted into the interval tree, with end point data being stored at the leafs, and intervals stored in the router nodes. We claim that the overhead of managing the tree grows linearly with respect to the number of objects in the scene at any one time. However, this cost is dependent upon the amount of information kept in the tree. Basically, as the tree becomes bigger, operations performed upon the tree slow down. The solution to this is to prune old information when it has been used. This is not a problem in the current small scenarios but would need attention if this technique was used to hold more data.

The cost of maintaining a static interval tree can be described as follows. Let $U \subseteq \mathbb{R}$ be an ordered finite set, $N = |U|$, and let $S$ be a set of $n$ intervals with left endpoints in $U$.

- An interval tree for $S$ uses space $O(n + N)$.
- An interval tree for $S$ of depth $O(\log N)$ can be constructed in time $O(N + n \log N \alpha(n))$.
- Intervals (with leaf endpoints in $U$) can be inserted into an interval tree of depth $O(\log N)$ in time $O(\log n + \log N)$. The same holds true for deletion.

Let $S$ be a set of intervals and let $I = [x_0, y_0]$ be a query interval. Let the answer $A = \{[x, y] \in S; [x, y] \cap [x_0, y_0] \neq \emptyset\}$ be the set of intervals in $S$ intersecting $I$. Then, given an interval tree of height $h$ for $S$, one can compute $A$ in time $O(h + |A|)$.

The cost of the dynamic interval tree is much the same as for the static case, indeed the same search mechanism is used in both types of interval tree. The differences that are present affect the time of insertions and deletions from the tree, since additional tree manipulation is required (for rebalancing, possible router node insertion, and possible leaf node insertion). The total cost of $n$ insertions or deletions into an initially empty tree is $O(n \log n)$. For proofs and further details, see Mehlhorn [157].
2.2 Modelling Paths

There are a number of ways of describing object paths, none of which are ideal at representing the movement of an object. An ideal path model should allow:

- Predictions to be made of where the object is likely to go next. This is to provide a spatio-temporal expectation of what may happen next, to help understand what is happening now.
- Path completion of a missing section.
- Comparisons with other object paths, e.g., do they cross, overlap, or split as shown in figure C.6.

These all require reasoning about an object’s spatiotemporal history.

2.2.1 Curvature Model

A well known approach for modelling space curves is the Serat-Frenet formulae (see Bruce and Giblin [25, pages 28-44] or O’Neill [178, pages 56-74] or Koenderink [123, pages 168-194] for details), which defines the curve in terms of its curvature $\kappa$ and its torsion $\tau$. In chapter 4 we simplified the representation of curves by using a linked-rod approximation to the curve swept out by an object, with measurements taken from the posebox centroid. An example of this is shown in figure 4.7, where we can interpret the linked rods as tangents.

2.2.2 Conduits

The representation of vehicle paths by curves that pass through their centroids does not provide a description of an object’s spatial extent and pose. To do this we use a conduit which represents the spatio-temporal data of an object as it is swept out in space and time. Conduits make the dynamic properties apparent, even in an inherently static figure on a page, for example, see figure C.7.
Figure C.8: Following. This shows the spatial separation.

Figure C.9: Overtaking. The spatial overlap in one dimension is not very useful.

**Representing Conduits** The basic 2D+t unit is a six sided shape with the top and bottom faces being poseboxes in the xy-plane but at different time values. Figure C.7 illustrates how a 3-cell, six 2-cells, twelve 1-cells and eight 0-cells are needed to describe one 2D+t unit. A better alternative would be to use six hyperplanes that are oriented to enclose the 3-cell. This allows each conduit unit to be described by the pair \((s, d)\), called "CU". In each CU, \(s\) is a six tuple representing values for top, front, left, right, back, bottom, such that each value is a coefficients \((1, -1)\) that denotes the side of the hyperplane that is used. Also in each CU, \(d\) is a further six tuple, where each slot holds a descriptor that points to the respective hyperplane. Günther [92] describes how efficient set operations can be performed on this class of representation. Even so, reasoning with conduits is not easy. The main problem concerns the development of a representation that allows comparisons with other object paths to be made.

**Reasoning with Conduits** Once we have generated the conduits, we have the problem of interpreting them. If they intersect then there is a likely collision or near miss, but intersections of conduits are unusual. We are interested in describing typical behaviour such as following and overtaking. Figures C.8 and C.9 illustrate these two cases with part (a) showing the objects are time \(t\) and the parts (b) and (c) showing the generated conduits. In this case they are from predictions, but they could equally well be from a sequence of posebox updates. Part (b) shows the overhead view and part (c) shows the side view. Following can be identified by ignoring the temporal axis, and combining the xy-planes (i.e., top view) of each conduit, and intersecting the results as shown in figure C.8(b). Overtaking can be identified by ignoring a spatial dimension as shown in figure C.9(c). The problem is that this spatial dimension is the 2D manifold, called the "rectifying developable", that fits the space curve of the linked-rod model. Mapping the
APPENDIX C. SPATIO-TEMPORAL MECHANISMS

conduits into such a manifold to perform such as test is difficult and has not been done here.

DISPLAY OF CONDUITS The display algorithm used to show the conduits is based upon Fuchs [80] Binary Space Partitioning tree (BSP) which takes advantage of the hyperplanes and polygons used in the representation of conduits.

2.2.3 PATH PREDICTION

Path prediction is used to predict the position of an object between two and four time steps in the future. We do not use longer predictions because of the accumulation of error. There are a number of approaches to path prediction.

Koenderink [123, page 193] describes an algorithm for curve generation that uses curvature ($\kappa$) and torsion ($\tau$) and the following three equations $T' = \kappa N$, $N' = -\kappa T + \tau B$, $B' = -\tau N$ obtained from the Serat-Frenet formulae to generate a curve that progresses at equal arc-length intervals. However, we require prediction related to time steps not arc-length. Also once we have obtained the value for curvature and torsion from path of an object, keeping curvature and torsion constant can produce unrealistic paths, for example:

- when $\kappa = 0$ the prediction produces a straight line.
- when $\tau = 0$ the prediction produces a planer curve.
- when $\kappa$ is a constant $> 0$ and $\tau = 0$ the prediction produces a circle.
- when $\kappa > 0$ and $\tau$ is a constant $\neq 0$ the prediction produces a circular helix.
- when $\kappa$ is constant, the prediction produces a cylindrical helix.

The typical result is a helix, which makes apparent any positional noise in the signal, particularly when an object is moving slowly.

Results from using a Kalman filter approach (see Bar-Shalom and Fortmann [15]) to generating path predictions are shown in figure C.11 which uses data from the occlusion scenario. Frame 45 shows following behaviour similar to figure C.8. Frame 102 shows the identification of a stationary object and prediction of an object leaving at the next exit. Frame 111 confirms the prediction of the leaving object, and also illustrates the problem of noise with the nearly stationary object. Frame 138 shows the prediction that the smaller vehicle is going to overtake the larger one. A clearer example of this is shown in figure C.12. These frames show the usefulness of performing predictions, however, the current implementation does not interpret the prediction generated and using predictions all the time seems wasteful. Recent work by Krozel [129] provides further details about this problem of predicting the motion of a vehicle, together with its connections to intelligent path planning. He proposes a solution that uses offline computed gradient direction information to reduce runtime costs. The objective of Krozel's work is the identification of each vehicle's goal, something which is likely to be outside the field-of-view and not addressed in the HIVIS-systems.

Object-based path predictions are not used in either of the HIVIS-based systems described here. The cost of generating a prediction, the problem of noise, and the cost of
interpreting the prediction once produced make this approach unattractive. To provide a less complex form of prediction we use the information held in SPATIAL-LAYOUT.

2.2.4 Region-based prediction

Figure C.13 shows the difference between fine grain object prediction and the coarse region based prediction. Region based prediction does not restrict the path prediction to the object shape, instead it uses a cone of expanding occupancy that is delimited by road region boundaries. This is to describe the area of the environment that can be reached by the object, taking account of environmental properties held in the spatial representation. This form of region-based path prediction has been used to develop a spatial-based scheduling algorithm (Tsang and Howarth [235]).

In the implementation used in HIVIS-WATCHER the region-based prediction uses the region-neighbour connectivity present in SPATIAL-LAYOUT. The prediction produced is more robust than that generated from object paths, and not affected by object position noise. However, it does ignore the speed of the object. Also, the choice of the fiducial size of region tessellation is important. This is because path predictions contain a temporal
component, and in region based predictions this temporal component is expressed by the amount of time it takes to travel through each region. If we have regions of similar size then the results from comparisons between two path predictions make more sense, because we have removed one temporal variable from consideration.

A simple example of the tessellation required is shown in figure C.14 which illustrates the path through a room that connects a main access corridor to some offices. More complex examples are shown by figures C.15 which shows three forms of region linking used to perform predictions using regions.

2.2.5 Traffic Flow Model

The traffic-flow model is intended to reflect recent typical behaviour of the scene objects. This extends the static spatial representation to include dynamic contextual information. Dynamic contextual information includes application domain data such as changing weather conditions and the actions of scene participants. For example, in the road-traffic domain a road user may have an accident, blocking a lane, or road works might alter the traffic flow. In the office domain, consider someone placing a large box in the middle of the typical path through the room. These actions break normal behaviour, and the model of typicality may need to be changed, so that for some temporal interval it can represent the altered traffic flow due to these obstacles. Once the abnormal situation is resolved we may have to learn the old state again, and until this is done strange predictions may be produced such as going around an area previously occupied by a broken down car (an example of this is given by Sutton's [224] blocking and shortcut problem).

The assumption is that the traffic flow model can be learnt from watching objects travel through the scene. At each time frame, we work out which region the object is in and store the object’s orientation and speed in a slot with the cell’s name. Once the run is complete, we find the average value of orientation and speed for each cell, which acts as an estimate. Figure C.16 shows the results from implementation of this simple algorithm. The results
can be used by a prediction program, such that given a position and an orientation, it can generate a typical path. In chapter 2 section 2.2.2 we described work by Mohnhapt and Neumann [161] that presents some exploratory research in this topic. An extension to this would be to use a hidden-Markov model to learn small paths from a given position. This would overcome the problem of averaging over two intersecting or diverging paths.

There is a link between learning the path prediction data and deriving the shape of the various spatial layout regions. For example, the shape of a turning region is based on information concerning the location where vehicles typically turn. This approach may prove useful as part of a knowledge acquisition phase that would be used to automatically generate the spatial layout region decomposition.
3 Spatial arrangements

Here we provide implementation details of the spatio-temporal operators introduced in sections 6.1.1 and 6.2.1 of chapter 4 which described the deictic primitive called "positional", and the gestalt primitive called "proximity".

3.1 Local object locations

Here we describe object based frame-of-reference for the observed scene objects. In the model used here an attended to object becomes the reference object, and we can only have one reference object at any one time, and to denote this we call the reference object self. The other located objects are called other. It is relatively easy to get a measurement of how far away the other object is from the half plane, making this approach attractive. In 2D a hyperplane is a line, so we can use a cross product to determine on which side of a given line, a given point lies. We will label the lines two end points $p_0$ and $p_2$, and label the point $p_1$. The distance $d$ is calculated by

$$d = \frac{(p_2 - p_0) \times (p_1 - p_0)}{|(p_2 - p_0)|}$$

If the result $d$ is negative then line $p_0 p_1$ is counterclock to line $p_0 p_2$. If $d$ is zero they are equal. Organising the oriented hyperplanes in a clockwise order gives the effect of inside and outside.

To explain this let us place a marker on a particular object and use the centroids of the other objects and the four hyperplanes that define the shape of the marked object. To find the distance of a centroid from the marked object we calculate the distance of a centroid from each of its four planes. The planes are orientated so that if the value is positive then the point is on the "inside", else it is on the "outside". I've called the two half-spaces these names to reflect how the four "insides" are composed to describe the marked object's spatial extent, i.e., if all the planes return a positive value then the centroid is inside the marked object. The distances are normal to the hyperplane, i.e., shortest distance of the point from the hyperplane. When the centroid produces an outside result on two planes we apply Pythagoras' Theorem to get the distance of the centroid from the hyperplanes place of intersection which is a corner on the vehicle (a lower dimensional face). To see how the method works consider figure C.17. The only problem with this approach is that we require stable descriptions of vehicle shape. Maybe adding a constraint that the distance between opposite corners should be the same (i.e., the internal chords should be of equal lengths).

Some of the ideas behind this are taken from Fuchs' description of BSP trees [80], Whitney's description of polyhedral chains [253] and Günther's application of the polyhedral chain model using hyperplanes [93].
If the result is negative then line p0 p1 is counterclockwise to line p0 p2, else if zero they are equal.

Function side-of-line (line p2)

```
let p0 := (first-point line); p1 := (second-point line);
ax := (x(p1) − x(p0)); ay := (y(p1) − y(p0));
bx := (x(p2) − x(p0)); by := (y(p2) − y(p0));
return (bx*ay − ax*by) / √(ax² + ay²)
```

Function workout-centroid-vehicle-position (shape point)

```
let Fd := (side-of-line (front shape) point);
Bd := (side-of-line (back shape) point);
Ld := (side-of-line (left shape) point);
Rd := (side-of-line (right shape) point);
isF := (Fd < 0); isB := (Bd < 0); isL := (Ld < 0); isR := (Rd < 0);
place := "" ; distance := 0;
if (isF ∧ isB ∧ isL ∧ isR) do place := "body" od
if else (¬ isF) ∧ isB ∧ isL ∧ isR) do place := "front" ; distance := Fd od
if else (isF ∧ ¬ isB) ∧ isL ∧ isR) do place := "back" ; distance := Bd od
if else (isF ∧ ¬ isB ∧ ¬ isL) ∧ isR) do place := "left" ; distance := Ld od
if else (isF ∧ isB ∧ ¬ isL ∧ ¬ isR)) do place := "right" ; distance := Rd od
if else (¬ isF) ∧ ¬ isB ∧ ¬ isL ∧ ¬ isR) do
  if (Fd > Ld) do place := "front" od else place := "front-left" od
distance := √(Fd² + Ld²) od
if else (isF ∧ ¬ isB ∧ ¬ isL ∧ ¬ isR) do
  if (Bd > Ld) do place := "back" od else place := "back-left" od
distance := √(Bd² + Ld²) od
if else (isF ∧ ¬ isB ∧ ¬ isL ∧ ¬ isR) do
  if (Bd > Rd) do place := "back" od else place := "back-right" od
distance := √(Bd² + Rd²) od
if else (¬ isF) ∧ isB ∧ ¬ isL ∧ ¬ isR) do
  if (Bd > Ld) do place := "back" od else place := "back-left" od
distance := √(Bd² + Ld²) od
else error
return (place, distance)
```

Table C.1: Positional feature algorithm.
3.2 Proximity

In chapter 6 we describe a model of proximity and here we provide details of how such a model can be implemented. We have modelled proximity as a field of graded intensity, and have assumed an analogy to the concentric gradient of Schöne [211, pages 30-32] that is used to describe how the intensity of a stimulus decreases gradually as the distance from its source increases. The proximity value increases as the object becomes progressively nearer, providing warnings about a possible collision and a measure of how proximate the other object is. We use the graded intensity values \([0, \ldots, 4]\) corresponding to the set \{not-near, nearby, close, very-close, touching\}, and to identify which objects are near enough to warrant a proximity measure we use an abound rejection test.

The gradient is implemented using the potential field method described by Latombe [135, pages 295-355]. We do not require a full implementation, we just use the repulsive potential which we apply to the selected object (as opposed to the obstacle regions in Latombe's description). For a given other object the function returns a "force" value related to the other objects proximity. The measure of proximity is described by:

\[
V_{rep}(x) = \begin{cases} 
\eta \left( \frac{\rho}{\rho_0} - 1 \right)^2 & \text{if } \rho(x) \leq \rho_0 \\
0 & \text{if } \rho(x) > \rho_0 
\end{cases}
\]

where \(\eta\) is a scale factor, \(\rho(x)\) denotes the Euclidean distance from point \(x\) to the other object, and \(\rho_0\) is the distance of influence of reference object. For the agent value space we use \(V_{agent} = \min([V_{rep}(x)], 4)\). Figure C.18 shows the values produced for \(V_{rep}(x)\) given \(\rho_0 = 10000\text{mm}, \eta = 1.5, \text{and } \zeta = 900\).
APPENDIX D

BAYESIAN NETWORKS

This appendix describes some implementation details and provides a brief tutorial on Bayesian networks. More details are given by Pearl [183] and Neapolitan [172]. We have used Bayesian networks in the implementation of HIVIS-WATCHER because they provide an attractive approach to modeling the uncertainty present in high-level reasoning about visual perception. Uncertainty arises in a number of forms such as occlusions, noise in the generation of the compact-encoding, and bias present in the perceiver due to its current surveillance task.

We begin with a description of some foundations, providing definitions of some basic terminology. This is followed by some implementation details concerning the inference algorithm, and a section providing an example of the "weights" mechanism.
1 Causal polytrees

The polytree algorithm of Kim and Pearl [120, 183] is a fundamental method for evidential reasoning in Bayesian networks. It provides exact solutions for singly connected networks using efficient, local computation. We begin by briefly describing what a causal polytree is, how conditional probability matrices are represented, and how local updating is performed by message passing. These foundations are then used to explain how to combine joint probabilities.

1.1 Computational approach

Definition D.1 (Pearl [183, page 176]) A Bayesian network is singly connected if no more than one path exists between any two nodes. Given a DAG = (V, E), then ∀u, v ∈ V there is at most one chain between u and v.

Definition D.2 (Pearl [183, page 150]) A Bayesian network is a causal tree if every node except the one called root has exactly one incoming link (i.e., one parent).

Definition D.3 (Pearl [183, page 176]) A Bayesian network is a causal polytree if any node in the network has multiple parents and the network is singly connected.

Alternative names to polytree are “singly connected network” and “generalised Chow tree” (see Chow and Liu [39]). Notice that a causal polytree is a more general form of causal tree and that they are both singly connected. The Bayesian networks in HIVIS-WATCHER are causal polytrees.

1.1.1 Conditional probabilities

Definition D.4 (Pearl [183, pages 150–151]) A fixed conditional probability matrix, M, qualifies each directed link X → Y, where each entry (x, y) in M is given by

\[ M_{y|x} \triangleq P(y | x) \triangleq P(Y = y | X = x) = \begin{bmatrix} P(y_1 | x_1) & \cdots & P(y_n | x_1) \\ \vdots & \ddots & \vdots \\ P(y_1 | x_m) & \cdots & P(y_n | x_m) \end{bmatrix} \]

The direction of the arrow designates X as the set of causal hypotheses and Y as the set of consequences or manifestation of these hypotheses. X is sometimes called the parent and Y the child.

A conditional probability matrix is associated with each “processing” node and is used to translate the values received from and sent to the parent nodes. This is easier in a causal tree than in a causal polytree because of the increase in parents. Each additional parent increases the dimension of the conditional probability matrix, i.e., \( \text{dim}(M) = 1 + \text{number-of-parents} \).
Graph structure is more important than the conditional probability matrices, which is fortunate since conditional probabilities typically need to be determined by hand. It is possible to learn a conditional probability matrix (see for example Cooper and Herkovits [45]) but this requires a large amount of data. The values in a conditional probability matrix sum to 1 along the y-direction in $M_{slc}$ given above. They tend to be selected either to reflect a considered distribution of how evidence is likely to cause some outcome, or to act as an identity by using a distribution similar to the Kronecker delta. An alternative to this numerical approach is to use qualitative Bayesian networks (Wellman [249]) which removes some of the difficulty associated with selecting "correct" values.

1.1.2 Message passing

The absence of loops in causal polytrees allows us to use local updating. In figure D.1 node $A$ has the set of parents $P = \{B, D\}$ and the set of children $C = \{X, Y\}$. We let $e$ be the total evidence obtained, $e_A$ be the evidence connected to $A$ through its children, $C$, and $e_A^\perp$ be the evidence connected to $A$ through its parents, $P$, so that $\text{BEL}(a_i) = \alpha P(e_A^\perp | a_i)P(a_i | e_A^\perp) = \alpha \lambda(a_i)\pi(a_i)$. The evidence $e_A^\perp$ and $e_A^\perp$ can be further decomposed into $e_A^\perp = \{e_{AX}^\perp, e_{AY}^\perp\}$ and $e_A^\perp = \{e_{BA}^\perp, e_{DA}^\perp\}$. Where $e_{AX}^\perp$ stands for evidence contained in the subnetwork on the head side of the link $A \rightarrow X$, and $e_{BA}^\perp$ stands for evidence contained in the subnetwork on the tail side of the link $B \rightarrow A$. Thus in this example the node $A$ partitions the graph into four subgraphs.

The message passing formula are $\pi_A(b_i) = P(b_i | e_{BA}^\perp)$ which provides causal support, and $\lambda_A(b_i) = P(e_{BA}^\perp | b_i)$ which provides diagnostic support. Note that $e_{BA}^\perp$ stands for the evidence contained in the subnetwork on the head side of the link $B \rightarrow A$, and that the link $B \rightarrow A$ partitions the graph into two parts.

The probability $P(A | B, D)$ relates the variable $A$ to its immediate causes, and that $B$ and $D$ are assumed to be a priori independent. When a common symptom is observed they become coupled forming an inter-cause relation.
1.1.3 Node Types

In the network structure there are different types of terminal and evidence nodes.

- **Anticipatory node** A leaf that has not been instantiated. The value of BEL should equal $\pi$, and $\lambda = (1, 1, \ldots, 1)$

- **Evidence node** A variable with instantiated value $\lambda(x) = P(x^- | x)$. If the $f^{th}$ value of $X$ is observed to be true, we set $\lambda(x) = (0, \ldots, 0, 1, 0, \ldots, 0)$ with 1 at the $f^{th}$ position.

- **Dummy node** A node $Y$ representing virtual or judgemental evidence bearing on $X$. We do not specify $\lambda(y)$ or $\pi(y)$ but instead post a $\lambda_y(x)$ message to $X$, where $\lambda_y(x) = \beta P(\text{Observation} | x)$, $\beta$ being a convenient constant making the message sum to 1.

- **Root node** The boundary condition for the root node is established by setting $\pi(\text{root})$ equal to the prior probability of the root variable $P(\text{root})$.

1.1.4 Combining Joint Probabilities

In section 1.1.1 we identified the problem of defining the conditional probability matrix $M$ for a node that has multiple parents. There are two basic approaches to solving this problem. The first is to define $M$ one n-dimensional entry at a time, and the other is to define $M$ in terms of 2D matrices that are combined to provide $M$. This second approach, which we will call “CJP”, is appealing because it reduces the amount of data that has to be specified and allows one 2D matrix definition to be used as itself as well as in different higher dimensional matrices, providing a consistency of the values used. This approach has been described by Patil et al. [182] (who called it “component summation”) and Kim and Pearl [120, 121]. They define an approximation of a higher-order conditional probability such as $P(A | B, D)$ in terms of lower order conditional probabilities $P(A | B)$ and $P(A | D)$ by using

$$P(a_i | b_k, d_l) = \alpha P(a_i | b_k)P(a_i | d_l)$$  \hspace{1cm} (D.1)

Unfortunately Neapolitan [172, pages 137–138] identifies a problem with this approach due to the assumptions of conditional independence. He provides an example where conflicts arise.

A separate problem is its lack of expressiveness. For example, the CJP is unable to accurately model the specification of the 3D matrix given in figure D.2, (which is based on equation D.2 from section 2). Part (a) of this figure illustrates three slices (move, $t+1$), (stat, $t+1$) and (unk, $t+1$) through the 3D conditional probability matrix that we would like to define. In comparison, part (b) illustrates a “best guess” of input to CJP. Note that the two 2D matrices are combined via equation D.1 to yield a 3D resultant matrix. There are clearly some problems with the CJP result. For example, the values for move and stat are difficult to separate. This limitation of expression effects the quality of the approximation produced by using the CJP approach, however, this does not prevent it from being useful as a tool for representing the “made-up” distribution used here. The identification of the loss of accuracy due to the CJP approach is a subject for future work.
Figure D.2: These are two different representations of an example developed from equation D.2: (a) gives the ideal 3D matrix in three slices,\(^2\) (b) gives the CJP input.

In HIVIS-WATCHER we use the function CJP of type \([M] \rightarrow M\) that takes a list of 2D conditional probability matrices and returns their higher-order composition by performing the computation described by equation D.1.

1.2 Implementation Details

1.2.1 The Inference Algorithm

Neapolitan [172, pages 238–240] provides a good description of probability propagation in singly connected networks and provides examples that allow the two parent case to be generalised to the multiple parent case.

\(^2\)Note that in (a) the empty squares are 0, and that the 1’s represent entries of equal probability. The 3D matrix constructed from these three slices would need scaling before it can be used as a conditional probability matrix, so that it complies with definition D.4.
The calculation of $\pi$-messages given by Pearl [183, page 170]

$$\pi_{a_k}(b_i) = \alpha \pi(b_i) \prod_{k \neq j} \lambda_{a_k}(b_j)$$

solves the divide by zero problem present in Neapolitan’s formula

$$\pi_A(b_j) = \frac{P'(b_j)}{\lambda_A(b_j)}$$

The problem of sending a $\lambda$-message is made easier by adapting Pearl’s [183, page 183] description to give

$$\lambda_{A}(b_j) = \alpha(\sum_{u_1, \ldots, u_n} P(a_i \mid u_1, \ldots, u_n) \prod_{k \neq j} \pi_A(u_k))\lambda(a_i)$$

where $(\sum_{u_1, \ldots, u_n} P(a_i \mid u_1, \ldots, u_n) \prod_{k \neq j} \pi_A(u_k))$ returns a 2D matrix that is used as a conditional probability matrix with $\lambda(a_i)$.

The conditional probability matrices are defined prior to runtime, and are held in global storage in an indexable data-structure called “CPMGS”, with links made from the Bayesian network to the relevant conditional probability matrix. Allocation of a conditional probability matrix to a node is determined by the types of the node’s parents. This information is used to index CPMGS, with the order of the parents has an injective mapping to the order of the dimensions in the conditional probability matrix, so that type and size information is correctly matched.

1.2.2 Multiplication and summation issues

We can speed-up multiplications involving conditional probability matrices by storing an additional list of matrix address representing those matrix address that hold a value greater than some minimum-threshold such as 0.01. This has no significant effect on the result since multiplying an intermediate value by zero makes no difference to the summed result. For example we list which of the $\sum a_i \times b_j a_i$ and $b_j$ addresses are worth using.

The summation over the number-of-parents is exponential in the number-of-parents and to address this Pearl [183, pages 183-194] proposes some approximation techniques for when there are more than five parents to a node. In the graphs generated by the DDN we do not often have parents with more than five parents and have not used any of these approximation techniques.
2 Weights

In chapter 5 we raised the issue of representing temporal durations. This is not obviously a problem, for example, it is easily addressed by the temporal logics described in appendix C which all have access to their underlying model of time, be it continuous (i.e., \( \mathbb{R} \)) or discrete. To represent this problem we might use something like:

\[
\forall u_1, u_2, c \quad ((u_2 > u_1 + \text{TYPICAL-WAIT-TIME}) \\
\land \text{TRUE}(u_1, u_2, \text{CAR-STOPPED}(c)) \\
\land \text{TRUE}(u_1, u_2, \text{CAR-STATIONARY}(c)) \\
\Rightarrow \text{TRUE}(u_2, u_2, \text{CAR-NOT-WAITING}(c))
\]

Which identifies when any car \( c \) is stationary for longer than the TYPICAL-WAIT-TIME, if this is the case than an error is raised to say that the car should no longer be considered to be waiting.

A different approach is taken by Nicholson and Brady [176] which can be used to monitor temporal durations such as those expressed in the example given above. Nicholson and Brady describe a mechanism called "weights", which can provide a limited history of some temporally changing object property such as motion. This approach uses a probabilistic method to provide a qualitative measure of time. In the description below we will let \( T \) be the current time and \( T + 1 \) be the next time-point, which is use to provide an expectation of what will happen next. We will use \( M_C(T) \) to describe the motion property of object \( c \) at time \( T \). \( M_C(T) \) has the ordered set of three vectors, with each component corresponding to an element from the domain \{stat, move, unk\}. Associated with \( M_C(T) \) is the weight \( W_C(T) \) which we will use to describe how long the object \( c \) has been stationary. The set \( W_C(T) \) corresponding to \( \{w_0, w_1, w_2, \ldots, w_{\text{max}}\} \) describes the finite number of weights that can be given values for node \( W_C \) at time \( T \), in this example, \( w_{\text{max}}-1 \) is the TYPICAL-WAIT-TIME. The nodes are connected as shown in figure D.3, and the probability distribution for the \( W_C(T) \) node (for a weight being incremented or decremented by 1, or staying the same) is:
\[
P(W_C(T + 1) = w_{\text{max}} | M_C(T) = \text{stat}, \ W_C(T) = w_{\text{max}}) = 1 \\
P(W_C(T + 1) = w_{n+1} | M_C(T) = \text{stat}, \ W_C(T) = w_n) = 1 \\
P(W_C(T + 1) = w_0 | M_C(T) = \text{move}, \ W_C(T) = w_0) = 1 \\
P(W_C(T + 1) = w_{n-1} | M_C(T) = \text{move}, \ W_C(T) = w_n) = 1 \\
P(W_C(T + 1) = w_0 | M_C(T) = \text{unk}, \ W_C(T) = w_n) = 1 \\
\] (D.2)

In this example the motion node, \( M \), provides information as to whether the object has moved, is stationary, or has unknown motion, by using the stat, move and unk states. If the object is stationary then the weight is incremented, unless the maximum weight has been reached. If the object moves then the weight is decremented, unless it is already at the minimum (i.e., \( w_0 \)). We also need to test when \( W_C(T + 1) = w_{\text{max}} \) to determine whether car \( c \) has been stationary for longer than the TYPICAL-WAIT-TIME. This mechanism is able to reason about a car that moves intermittently, say one that edges forward at a junction. The temporal logic version might reset itself, whilst the probabilistic model would be able to provide a more realistic solution with small fluctuation in the likelihood of interpretation.
APPENDIX E

OPERATIONAL SEMANTICS FOR MACNET

The main reason for this appendix is to describe my interpretation of Agre and Chapman’s circuit-based form of knowledge representation. The descriptions given here may not be the same as those of Agre and Chapman, because although Chapman [34] provides a syntax and a semantics given in an informal manner with examples, there is ambiguity in how what is described might be implemented and what it means. The description here is intended to bridge this gap by presenting my interpretation of MACNET. The semantic description given here is based upon VDL [9, 180].

This is not the only approach that could have been taken because MACNET has similarities to rule based systems, and default logic. Agre [2] discusses the differences to rule based systems which basically come down to MACNET having more complex conflict-resolution for parallel rule satisfaction and selection.

MACNET also seems similar to default logic\(^1\) (Reiter [196]) with the elements of the MACNET language mapping onto default rule form as follows: “propose” being the prerequisite, the “conditions” and “overriders” being the justification, and the “arbiter” value allocation being the consequent. Although this mapping would have fitted well in the AI field, the proofs of equivalence would be difficult and we do not follow this route here.

Having implemented MACNET and used it for developing deictic systems, it may be surprising to find that there is little in its definition that makes it a necessary element in the development of future deictic implementations. Wires and circuits are not needed for deictic reasoning. Their presence, however, limits the options available, guiding the process of implementation towards something that may support the deictic viewpoint. Most of the complexity in MACNET supports ways of taking environmental conditions into account when choosing between alternative courses of action.

The implementation of MACNET does not require the use of Lisp, although it does make some aspects of the implementation easier. Also there is no reason that the “linear executable form” from the gate language could not be translated to a host architecture’s machine code, should faster execution be required. After all, when you get the the bottom

\(^1\)A default inference rule in default logic is written in the form:

\[
\frac{\alpha(x): \beta(x)}{\gamma(x)}
\]

where \(\alpha(x)\), \(\beta(x)\) and \(\gamma(x)\) are well-formed formula called the “prerequisite”, the “justification” and the “consequent” of the default respectively. The interpretation of this rule is as follows. If \(\alpha(x)\) is known, and \(\beta(x)\) is consistent with what is known, then \(\gamma(x)\) may be concluded.

274
of it, what we describe here is a logic simulator that executes a given network of combinatorial logic gates.

In the description that follows we first describe the gate language and then describe how the MACNET language is defined in terms of the gate language. Each of these descriptions takes two sections with the first providing an overview and the second giving the formal details.
Appendix E. Operational Semantics for MACNET

(defun make-adder (*a* *b* *cin* *cout* *sum*)
  (labels
    ((halfadder (in1 in2 out1 out2)
      (let ((x (andg in1 in2)))
        (sequ (set-wire! out1 (andg (invert x) (org in1 in2)))
              (set-wire! out2 x)))))
    (let (((w1 (gentemp "WIRE")) (w2 (gentemp "WIRE")))
           (w3 (gentemp "WIRE")) (w4 (gentemp "WIRE")))
      (sequ (halfadder (interface-node *b*) (interface-node *cin*) w1 w2)
            (halfadder (interface-node *a*) (interface-wire w1) w3 w4)
            (set-node! *sum* (interface-wire w3))
            (set-node! *cout* (org (interface-wire w4)
                                    (interface-wire w2))))))

Figure E.1: Adder circuit.

Figure E.2: A half-adder.

Figure E.3: A full adder.

1 GATES

The gate description language provides the user with a collection of Lisp functions for constructing Lisp structures. These functions can be treated like usual Lisp objects, although the result of their composition needs to maintain the gate language syntax. Figure E.1 provides an example of an adder, showing how this flexibility allows sub-circuit forms to be assigned to variables that can then be used in more than one place. The important gates in this example are andg, org and invert, which correspond to the usual combinatorial logic meanings of ∧, ∨ and ¬ respectively. The general gate syntax is (foo g &rest input) which defines the gate foo. Each gate takes one or more input wires depending upon the gate's meaning and all gates have a single output wire. Figures E.2 and E.3 express the code in pictorial term, making clearer the relationships between the components and sub-circuits. In figure E.3 the labels A and B represent the binary bits at corresponding positions in the two numbers to be added, and Cin is the carry bit from the addition one place to the right. The circuit generates SUM, which is the sum bit in the corresponding position, and Cout, which is the carry bit to be propagated to the left. The tree structure that is created by the Lisp functions is shown in figure E.4, even for this small example, it is quite large, and not very illuminating. However, it does give some idea of how the gate language can be composed to produce a tree of gates.

In addition to the andg, org and invert gates already described above, other gates include (interface-node input-symbol) which "reads" the current value held by the specified input symbol. (set-node! output-symbol input) which "writes" the value of the
input wire to the specified output symbol. (interface-wire name) which accesses the named wire, this wire acts like a variable holding its set value during the clock tick. At compile time a check is made to ensure that the wire is set before it is used via the (set-wire! name input) gate which sets the named wire to a new value. The jobs performed by these node and wire functions could be combined into one pair of functions, however keeping them distinct clarifies the interval wire operations from the node links of the circuit with the external world. A (latch input) gate that provides a one clock tick delay function which is used to supply the value from the previous clock tick and hold the current value. (saym &rest input) is used to print warnings and error messages. To allow switching between inputs there are three gate forms based on an ifm primitive. An if-then form (ifm test input), an if-then-else form (ifm test input1 input2) and also a cond macro that has the usual Lisp cond syntax and expands into a collection of nested ifm gates. The cond is to make specifying complicated ifm's more understandable and is used in the construction of arbiters. In the adder example we also use (sequ &rest input) to collect together a sequence of subtrees written in the gate language. The gate sequ acts like a “progn” in Lisp, allowing sequential execution, but does not return any value.

Standard theorem proving techniques are not used to evaluate a circuit, instead it is run by a program based on a Digital LSI Design simulator [230]. The simulator is based upon the observation that a circuit of gates forms a tree structure that can be post-order tree-walked to evaluate the values held at each node, starting from the root (or final output) gate. This approach is fine for a single network execution however, there is a more efficient way of evaluating a circuit tree that is to be repeatedly run with new leaf (or input gate) values.

This second method separates out from the “evaluation phase” work that can be done as a pre-process, which we will call, “network analysis”. Network analysis performs a tree-walk, from the output node, not descending any further when an input node or dead end is reached. During the tree-walk all switches are assumed to be on, since the tree-walk is performed before any node values are calculated. During this tree-walk the linear runtime structure, called the “code-array”, is constructed, which holds the gate nodes in post-order, ensuring that when we iterate through the code-array beginning from address 0, all arguments for each gate node are evaluated before they are used.

This constructs our network ready for simulation. A simple approach to running the simulator is to have two node-value arrays; one to hold the current values of each node, the other to collect new values as they are computed. Each node is assigned an index which can be used to access its current value in the first array or to store its new value in the second array. The algorithm for this is:

1. For each input node, set its current-value array entry to the designated input value.

2. Execute the simulation subroutine. This fills the new-value array.

3. Compare the current-value and new-value arrays. If their contents are identical the network has settled and the simulation step is over. Otherwise copy new-value array to old, and goto step 1.
Figure E.4: The tree structure formed from the adder example. Note that for simplicity the slots for LEVEL (and also for OUTPUT and RESULT from the gates set-wire! and set-node! respectively) have been removed, because they are not used until we create the linear form.
This method can be simplified by taking advantage of the post-order present in code-array. Instead of using two value arrays, we can make do with a single value array, because the values of a node's inputs are calculated before the value of the node itself is calculated. The new algorithm, called Cascade is:

1. Set counter i to 0.
2. If i > length of array, exit
3. Execute the simulation subroutine for node[i] using its specified input addresses from the value-array (or elsewhere). Put the result in value-array[i].

Terman [230] describes a more complex version of the Cascade algorithm that can also cope with loops such as would be present in a flip-flop however, the gate language does not need this form of feedback, allowing us to ignore such issues.

Having outlined an informal description of the gate language, we will now present a formal description.
2 Gate Language Details

We will first outline the syntax of the gate language using a BNF that has been sugared to make the gate names prominent. The sugaring translates a function call of the form \( f[x_1; \ldots; x_n] \) to \( (f \ x_1^* \ldots x_n^*) \) where the star denotes the translation of the appropriate item.

\[
\begin{align*}
\lt wire \gt &::= \lt gate\ machine\gt \\
\lt atomic \gt &::= \lt integer \gt | \lt boolean \gt | \lt Lisp\ keyword \gt | \lt Lisp\ symbol \gt \\
\lt input \gt &::= \lt Lisp\ symbol \gt \\
\lt output \gt &::= \lt Lisp\ symbol \gt \\
\lt gate\ machine \gt &::= \text{seq} [ \lt wire \gt; \ldots; \lt wire \gt] \\
\ &\quad | \text{andg} [ \lt wire \gt; \ldots; \lt wire \gt] \\
\ &\quad | \text{invert} [ \lt wire \gt] \\
\ &\quad | \text{constant} [ \lt atomic \gt] \\
\ &\quad | \text{set-node!} [ \lt output \gt; \lt wire \gt] \\
\ &\quad | \text{interface-node} [ \lt input \gt] \\
\ &\quad | \text{interface-wire} [ \lt atomic \gt] \\
\ &\quad | \text{set-wire!} [ \lt atomic \gt; \lt wire \gt] \\
\ &\quad | \text{saym} [ \lt string \gt; \ldots; \lt string \gt] \\
\ &\quad | \text{ifm} [ \lt wire \gt; \lt wire \gt] \\
\ &\quad | \text{ifm} [ \lt wire \gt; \lt wire \gt; \lt wire \gt] \\
\ &\quad | \text{gtm} [ \lt wire \gt; \lt wire \gt] \\
\ &\quad | \text{lmt} [ \lt wire \gt; \lt wire \gt] \\
\ &\quad | \text{eqm} [ \lt wire \gt; \lt wire \gt]
\end{align*}
\]

The BNF notation defines the gate language syntax and the text so far has given this syntax an informal semantics. We will now define a more formal semantics of the gate language, using an operational approach where meaning is described in terms of abstract machines with discrete "states" and more-or-less explicit sequences of computational operations. The elementary objects are the values given to the circuit, these leaf values enter the circuit via gates such as interface-nodes, constant or interface-wires. The composite objects include the other gates, such as the output gate set-node!.

This abstract syntax defines the predicate \( P \) (prefixed by "is-") in terms of further predicates and selectors (prefixed by "s-"), i.e., \( P = (s_1 : P_1, s_2 : P_2, \ldots) \). Each definition can be considered to form a tree structure, with selectors as the branch names. The selectors we will use here include \text{s-gate-name} for the name of the gate AND, OR, etc, and \text{s-input} for the input wire to the gate (multiple inputs have subscripts). Further details about the abstract syntax used here are given in [9]. The following abstract syntax is used to decompose the gate machines according to their various arities:

\[
\begin{align*}
\text{is-gate} &= ( \text{is-unary-gate} V \text{is-binary-gate} V \text{is-ternary-gate} V \text{is-nary-gate} ) \\
\text{is-unary-gate} &= ( \text{s-input} : \text{is-gate}, \text{s-gate-name} : \text{is-latch} V \text{is-invert} ) \\
\text{is-unary-atomic} &= ( \text{s-input} : \text{is-atomic}, \text{s-gate-name} : \text{is-constant} V \text{is-interface-wire} ) \\
\text{is-unary-input} &= ( \text{s-input} : \text{is-input}, \text{s-gate-name} : \text{is-interface-node} ) \\
\text{is-binary-gate} &= ( \text{s-input}_1 : \text{is-gate}, \text{s-input}_2 : \text{is-gate}, \text{s-gate-name} : \text{is-ifm} V \text{is-gtm} V \text{is-ltm} V \text{is-eqm} )
\end{align*}
\]
is-binary-atomic-gate = ( <s-input₁ : is-atomic>, <s-input₂ : is-gate>,
                 <s-gate-name : is-set-wire! > )

is-binary-output-gate = ( <s-input₁ : is-output>, <s-input₂ : is-gate>,
                 <s-gate-name : is-set-node! > )

is-ternary-gate = ( <s-input₁ : is-gate>, <s-input₂ : is-gate>,
                 <s-input₃ : is-gate>, <s-gate-name : is-ifm > )

is-nary-gate = ( <s-input₁ : is-gate>, ..., <s-inputₙ : is-gate>,
                 <s-gate-name : is-org V is-andg V is-sequ >), <s-arity: n > )

is-nary-string = ( <s-input₁ : is-gate>, ..., <s-inputₙ : is-string>,
                 <s-gate-name : is-saym >), <s-arity: n > )

This has described a tree like abstract syntax that provides a good intuitive match to
the BNF syntax above, with the value of n in is-nary-gate obtained from the number
of inputs supplied. When a gate language circuit is given as input to network analysis
it is a larger tree made up of these abstract syntax subtrees. As already described we
will not execute over this larger tree, instead we will translate it into a linear structure
called net. This translation, called network analysis, requires some further abstract syntax
definitions.

is-net = ( <s-code : is-code>, <s-dim : is-integer>,
                 <s-wires : is-entrystore >), <s-result : is-entrystore >,
                 <s-latch-dim : is-integer > )

is-code = ( <s-addr₁ : is-addr>, ..., <s-addrₙ : is-addr > )

is-addr = ( <s-gate : is-gate > )

is-entrystore = ( <s-entry₁ : is-atomic>, ..., <s-entryₙ : is-atomic>,
                 <s-dim : is-integer > )

This has defined net, also shown in figure E.5, which is to hold the linear structure and
accompanying details. Once we have created net we will then wish to run it. During run-
time execution we use a runtime environment, enu, to hold input, result and intermediate
values, this has the following abstract syntax:
is-env = ( <s-value-array> : is-entrystore>,
            <s-latch-array> : is-entrystore>, <s-result-array> : is-entrystore> )

The above abstract syntax is used is both the following stages of network analysis and runtime execution. First let us describe what an instruction definition is. A mutation operator, \( \mu \), is used to alter the structure of an object defined by the various selectors used to describe its form. The selectors used here were defined using abstract syntax, part of which has defined the tree structure in figure E.5. The \( \mu \)-operator can be used for inserting, removing or altering parts of an already existing object or for creating a new object by operating upon, \( \Omega \), the null object. There are two cases worth describing here, (1) an object, \( A \), can be created via \( A \leftarrow \mu(\Omega; <S; \mathcal{P}>), \) which can be visualised as an object, \( A \), with a single branch for selector \( S \) with predicate \( \mathcal{P}. \) (2) \( \mu(B; <\chi; C>), \) where \( \chi \) is a selector and \( B \) and \( C \) are objects. Depending on the original composition of \( A \), this \( \mu \)-operator has one of three effects:

- if \( B \) contains no \( \chi \)-selector then a \( \chi \)-selector whose object is \( C \) is added to \( B \)
- if \( B \) does contain a \( \chi \)-selector, this is removed and replaced by \( C \)
- if \( C = \Omega \) the \( \chi \)-selector of \( B \) is removed

The instruction definition uses a Lisp cond like syntax to select the first predicate that is true. It takes the form

\[
\text{in } (a_1, a_2, \ldots, a_n) = \\
\mathcal{P}_1 \rightarrow \text{group}_1 \\
\mathcal{P}_2 \rightarrow \text{group}_2 \\
\vdots \\
\mathcal{P}_n \rightarrow \text{group}_n
\]

Here (1) in denotes the instruction name. (2) \( a_1, a_2, \ldots, a_n \) denote the parameters passed to the instruction. (3) Each group is a set of instructions. (4) \( \mathcal{P}_i \) is a predicate. The truth of \( \mathcal{P}_i \) implying that \( \text{group}_i \) may be selected for execution. The predicates are examined in order \( \mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n \) with the first predicate that is true producing the group to be executed.

An instruction of the form

\[
\text{in } (a_1, a_2, \ldots, a_n) = \\
T \rightarrow \text{group}
\]

is, for simplicity, replaced by

\[
\text{in } (a_1, a_2, \ldots, a_n) = \\
\text{group}
\]

The groups themselves are sequences of instructions representing a macro-expansion which generally indicates an application of the \( \mu \)-operator. The instructions given here are executed in the order implied\(^2\), i.e., \{ instruction\(_1; \) instruction\(_2; \ldots; \) instruction\(_n \} \), and results are returned by using the PASS component.

\(^2\)Note that this differs from standard VDL, see [9].
Having outlined the basic details of the approach to be taken, let us now consider how network analysis is to be carried out. We are performing a post order descent of the tree starting from the root. We do network analysis of the input to each gate which provides the set addresses, \( A \), at which the values for each input resides. Note that given each address \( \alpha \in N \) and that addresses are allocated in increasing order, for any "current" address, \( i \), the following is true (\( \forall \alpha (\alpha \in A \text{ and } \alpha < i) \)).

\[
\text{do-network-analysis (machine) =}
\{ \text{net} \leftarrow \text{make-net-structure};
\text{network-analysis(net, machine);}
\text{PASS} \leftarrow \text{net} \}
\]

\[
\text{make-net-structure ( ) =}
\{ \mu(\Omega ; <s-code: \emptyset>, <s-dim: 0>, <s-wires: \text{make-entry-store}(mazlen)>,
< s-result : \text{make-entry-store}(mazlen) >, < s-latch-dim : 0 > ) \}
\]

\[
\text{network-analysis (net, machine) =}
\text{is-unary-gate(machine) } \rightarrow \text{network-analysis-unary-gate(net, machine)}
\]

\[
\text{is-unary-atomic(machine) } \rightarrow \text{network-analysis-unary-atomic(net, machine)}
\]

\[
\text{is-unary-input(machine) } \rightarrow \text{network-analysis-unary-input(net, machine)}
\]

\[
\text{is-binary-gate(machine) } \rightarrow \text{network-analysis-binary-gate(net, machine)}
\]

\[
\text{is-binary-atomic-gate(machine) } \rightarrow 
\text{network-analysis-binary-atomic-gate(net, machine)}
\]

\[
\text{is-binary-output-gate(machine) } \rightarrow 
\text{network-analysis-binary-output-gate(net, machine)}
\]

\[
\text{is-ternary-gate(machine) } \rightarrow \text{network-analysis-ternary-gate(net, machine)}
\]

\[
\text{is-nary-gate(machine) } \rightarrow \text{network-analysis-nary-gate(net, machine)}
\]

\[
\text{is-nary-string(machine) } \rightarrow \text{network-analysis-nary-string(net, machine)}
\]

\text{network-analysis} is the main point of recursion in the network analysis. The \( \text{net} \) formed by \text{make-net-structure} should use a adjustable entry-store since at this stage we do not know the length of \( \text{mazlen} \). An alternative would be to obtain these values via a first pass of the tree-structure of gates. We will now go through each of the gate functions as grouped together by their arities.

\[
\text{network-analysis-unary-gate (net, machine) =}
\text{is-latch(machine) } \rightarrow 
\{ \text{address} \leftarrow \text{network-analysis(net, s-input(machine))};
\text{P} = (<s-input: address>);
\text{<s-latch-addr: new-latch-addr(net)>};
\text{store(net, s-gate-name(machine), P)}) \}
\]

\[
\text{is-invert(machine) } \rightarrow 
\{ \text{address} \leftarrow \text{network-analysis(net, s-input(machine))};
\text{P} = (<s-input: address>);
\text{store(net, s-gate-name(machine), P)}) \}
network-analysis-unary-atomic \( (net, machine) = \)

\( \text{is-constant}(machine) \rightarrow \)
\( \{ \mathcal{P} = \langle s\text{-input} : s\text{-input}(machine) >; \)
\( \quad \text{store}(net, s\text{-gate-name}(machine), \mathcal{P}) \} \)

\( \text{is-interface-wire}(machine) \rightarrow \)
\( \{ \text{name} \leftarrow s\text{-input}(machine); \)
\( \quad \text{address} \leftarrow \text{get-wire-address}(net, name); \)
\( \quad \text{PASS} \leftarrow \text{address} \} \)

network-analysis-unary-input \( (net, machine) = \)

\( \text{is-interface-node}(machine) \rightarrow \)
\( \{ \text{name} \leftarrow s\text{-input}(machine); \)
\( \quad \text{address} \leftarrow \text{get-interface-address}(net, name); \)
\( \quad \text{PASS} \leftarrow \text{address} \} \)

network-analysis-binary-gate \( (net, machine) = \)

\( \text{is-ifm}(machine) \rightarrow \)
\( \{ \text{test} \leftarrow \text{network-analysis}(net, s\text{-input}_1(machine)); \)
\( \quad \text{jump}_1 \leftarrow \text{store}(net, \text{RESERVE}, \emptyset); \)
\( \quad \text{network-analysis}(net, s\text{-input}_2(machine)); \)
\( \quad \text{jump}_2 \leftarrow \text{store}(net, \text{is-noop}, \emptyset); \)
\( \quad \mathcal{P} = \langle s\text{-input}_1 : \text{test}, \langle s\text{-input}_2 : \text{jump}_2 >; \)
\( \quad \text{store-at-index}((\text{jump}_1, net, s\text{-gate-name}(machine), \mathcal{P}) \} \)

\( T \rightarrow \{ \text{address}_1 \leftarrow \text{network-analysis}(net, s\text{-input}_1(machine)); \)
\( \quad \text{address}_2 \leftarrow \text{network-analysis}(net, s\text{-input}_2(machine)); \)
\( \quad \mathcal{P} = \langle s\text{-input}_1 : \text{address}_1, \langle s\text{-input}_2 : \text{address}_2 >; \)
\( \quad \text{store}(net, s\text{-gate-name}(machine), \mathcal{P}) \} \)

network-analysis-binary-atomic-gate \( (net, machine) = \)

\( \text{is-set-wire!}(machine) \rightarrow \)
\( \{ \text{name} \leftarrow s\text{-input}_1(machine); \)
\( \quad \text{inaddr} \leftarrow \text{network-analysis}(net, s\text{-input}_2(machine)); \)
\( \quad \text{node} \leftarrow \text{store}(net, \text{RESERVE}, \emptyset); \)
\( \quad \text{save-wire-address}(net, name, node); \)
\( \quad \text{outaddr} \leftarrow \text{get-wire-address}(net, name); \)
\( \quad \mathcal{P} = \langle s\text{-input} : \text{inaddr}, \langle s\text{-output} : \text{outaddr} >; \)
\( \quad \text{store-at-index}(\text{node}, net, s\text{-gate-name}(machine), \mathcal{P}) \} \)

network-analysis-binary-output-gate \( (net, machine) = \)

\( \text{is-set-node!}(machine) \rightarrow \)
\( \{ \text{name} \leftarrow s\text{-input}_1(machine); \)
\( \quad \text{inaddr} \leftarrow \text{network-analysis}(net, s\text{-input}_2(machine)); \)
\( \quad \mathcal{P} = \langle s\text{-input} : \text{inaddr}, \langle s\text{-result} : \text{get-result-address}(name) >; \)
\( \quad \text{store}(net, s\text{-gate-name}(machine), \mathcal{P}) \} \)

network-analysis-ternary-gate \( (net, machine) = \)

\( \text{is-ifm}(machine) \rightarrow \)
\( \{ \text{test} \leftarrow \text{network-analysis}(net, s\text{-input}_1(machine)); \)
\( \quad \text{jump}_1 \leftarrow \text{store}(net, \text{RESERVE}, \emptyset); \)
\( \quad \text{network-analysis}(net, s\text{-input}_3(machine)); \)
\begin{verbatim}
jump_2 \leftarrow \text{store}(\text{net}, \text{RESERVE}, \emptyset);
\text{network-analysis}(\text{net}, s\text{-input}_2(m\text{achine})));
jump_3 \leftarrow \text{store}(\text{net}, \text{RESERVE}, \emptyset);
\mathcal{P} = \langle s\text{-input}_1 : \text{test} >, s\text{-input}_2 : \text{jump}_2 >);\n\text{store-at-index}(\text{jump}_1, \text{net}, \text{s\text{-gate-name}(machine)}, \mathcal{P});
\mathcal{P} = \langle s\text{-input} : \text{jump}_3 >);
\text{store-at-index}(\text{jump}_2, \text{net}, \text{is\text{-jump}}, \mathcal{P});
\text{store-at-index}(\text{jump}_3, \text{net}, \text{is\text{-noop}}, \emptyset)\}
\end{verbatim}

\textbf{network-analysis-nary-gate} (\text{net}, \text{machine}) =
\begin{verbatim}
is-seq(m\text{achine}) \rightarrow
\begin{cases}
\text{addresslist} \leftarrow \emptyset; \\
n\text{ary\text{-loop}(machine, 0, addresslist)}
\end{cases}
\end{verbatim}
\begin{verbatim}
T \rightarrow \begin{cases}
\text{addresslist} \leftarrow \emptyset; \\
\text{addresslist} \leftarrow n\text{ary\text{-loop}(machine, 0, addresslist)} \\
\text{store}(\text{net}, \text{s\text{-gate-name(}machine\text{)}, \text{addresslist})}
\end{cases}
\end{verbatim}

\textbf{network-analysis-nary-string} (\text{net}, \text{machine}) =
\begin{verbatim}
is-saym(m\text{achine}) \rightarrow
\begin{cases}
\text{STR} \leftarrow "\text{"}; \\
\text{STR} \leftarrow \text{saym-loop(m\text{achine}, 0, STR}); \\
\mathcal{P} = \langle s\text{-input} : \text{STR} >)
\end{cases}
\end{verbatim}
\begin{verbatim}
\text{store}(\text{net}, \text{s\text{-gate-name(}machine\text{)}, \mathcal{P})}
\end{verbatim}

This has described the main elements of network analysis, which has the task of building a linear structure called \text{net}. The network analysis process has used the following lower-level functions, that perform storage and access operations upon the \text{net} so that it can be constructed. We begin with the creation of an entry store, in an implementation this could be an array or a hash-table.

\textbf{make-entry-store} (limit) =
\begin{verbatim}
\begin{cases}
\mathcal{ES} \leftarrow \mu(\Omega ; s\text{-dim} : 0 >); \\
\text{makeES-loop(ES, 0, limit)}; \\
\text{PASS} \leftarrow \mathcal{ES}
\end{cases}
\end{verbatim}

There are two versions of the store function which are used to place a gate in the linear structure \text{s\text{-code}. The more general one, called \text{store}, will create a new address and store the given \text{gate} at the new address. \text{store-at-index} will just store a \text{gate} at the specified address, \text{T}.

\textbf{store} (\text{net}, \text{g\text{ate},input}) =
\begin{verbatim}
\begin{cases}
\mathcal{T} \leftarrow \text{new-address(}net\text{)}; \\
\text{store-at-index(T, net, gate, input)}; \\
\text{PASS} \leftarrow \mathcal{T}
\end{cases}
\end{verbatim}

\textbf{store-at-index} (\text{T}, \text{net}, \text{gate, input-predicate}) =
\begin{verbatim}
\begin{cases}
\mathcal{S}_\text{g\text{ate}} \leftarrow \text{s\text{-gate \circ s\text{-addr}}(T) \circ s\text{-code}}; \\
\mu(\text{net}; \langle s\text{-gate-name \circ S}_\text{g\text{ate}} : \text{gate} >); \\
\text{store-loop(}net, \mathcal{S}_\text{g\text{ate}}, \text{input-predicate})
\end{cases}
\end{verbatim}
Next, we describe the four iteration functions that were used in the definitions above. These definitions use two infix operators: @ = string concatenate and ++ = list append (these are equivalent to the definitions in [9, page 120]).

\[
\text{saym-loop} \ (\text{machine}, n, \text{STR}) = \\
\quad n > \text{s-arity} \ (\text{machine}) \rightarrow \{ \ \text{PASS} \leftarrow \text{STR} \ \} \\
\quad T \rightarrow \{ \ \text{STR} \leftarrow \text{STR} \circ \text{s-input}_i(\text{machine}); \\
\quad \quad i \leftarrow (i + 1); \\
\quad \quad \text{PASS} \leftarrow \text{saym-loop} \ (\text{machine}, i, \text{STR}) \ \} \\
\text{narym-loop} \ (\text{machine}, n, \text{addresslist}) = \\
\quad n > \text{s-arity} \ (\text{machine}) \rightarrow \{ \ \text{PASS} \leftarrow \text{addresslist} \ \} \\
\quad T \rightarrow \{ \ \text{address} \leftarrow \text{network-analysis} (\text{net}, \text{s-input}_i(\text{machine})); \\
\quad \quad \text{addresslist} \leftarrow \text{addresslist} \circ \langle \text{s-input}_i : \text{address} \rangle ; \\
\quad \quad i \leftarrow (i + 1); \\
\quad \quad \text{PASS} \leftarrow \text{narym-loop} \ (\text{machine}, i, \text{addresslist}) \ \} \\
\text{makeES-loop} \ (E_S, i, \text{limit}) = \\
\quad i > \text{limit} \rightarrow \text{null} \\
\quad T \rightarrow \{ \ \mu(E_S; \langle s-\text{entry}_i : \emptyset \rangle); \\
\quad \quad i \leftarrow (i + 1); \\
\quad \quad \text{makeES-loop} \ (E_S, i, \text{limit}) \ \} \\
\text{store-loop} \ (\text{net}, S_0, \text{input-predicate}) = \\
\quad \text{is}\emptyset(\text{input-predicate}) \rightarrow \text{null} \\
\quad T \rightarrow \{ \ \langle S; P \rangle \leftarrow \text{head} (\text{input-predicate}); \\
\quad \quad \mu(\text{net}; \langle S \circ S_0 : P \rangle); \\
\quad \quad \text{store-loop} \ (\text{net}, S_0, \text{tail} (\text{input-predicate})) \ \} \\
\]

Finally, the network analysis functions for getting and creating storage addresses in the net structure that forms the static runtime environment.

\[
\text{get-result-address} \ (\text{net}, \text{name}) = \\
\quad \{ \ I \leftarrow \text{s-entry}_\text{name} \circ \text{s-result} (\text{net}); \\
\quad \quad I \leftarrow \text{present-result-address} (\text{net}, \text{name}, I); \\
\quad \quad \text{PASS} \leftarrow I \ \} \\
\text{present-result-address} \ (\text{net}, \text{name}, I) = \\
\quad I = \emptyset \rightarrow \\
\quad \{ \ I \leftarrow \text{s-dim} \circ \text{s-result} (\text{net}); \\
\quad \quad \mu(\text{net}; \langle \text{s-entry}_\text{name} \circ \text{s-result} : I \rangle); \\
\quad \quad \mu(\text{net}; \langle \text{s-dim} \circ \text{s-result} : (1 + I) \rangle); \\
\quad \quad \text{PASS} \leftarrow I \ \} \\
\quad T \rightarrow \{ \ \text{PASS} \leftarrow I \ \} \\
\text{new-latch-addr} \ (\text{net}) = \\
\quad \{ \ \mu(\text{net}; \langle \text{s-latch-dim} : (1 + \text{s-latch-dim} (\text{net}) \rangle) \ \} \\
\text{get-wire-address} \ (\text{net}, \text{name}) = \\
\quad \{ \ \text{PASS} \leftarrow \text{s-entry}_\text{name} \circ \text{s-wires} (\text{net}) \ \} \\
\text{save-wire-address} \ (\text{net}, \text{name}, \text{node}) = \\
\quad \text{get-wire-address} (\text{net}, \text{name}) \rightarrow \text{null}
\[ T \rightarrow \{ \mu(\text{net}; <\text{s-entry name} \circ \text{s-wires} : \text{node}>) \} \]

**new-address** (net) =
\[
\{ \mathcal{D} \leftarrow \text{s-dim}(\text{net}); \\
\mu(\text{net}; <\text{s-dim} : (\mathcal{D} + 1)>) ; \\
\text{PASS} \leftarrow \mathcal{D} \}
\]

This completes the instruction definition giving as operational semantics for network analysis. All that remains is to describe how the net built by the network analysis can be executed at runtime using **exec-net**. This is the abstract machine which has the general form \( \xi_{i+1} = \Lambda(\xi_i) \), where \( \xi_0, \xi_1, \xi_2, \ldots \) are a sequence of states and \( \Lambda \) is the state transition, a partial function. In the abstract machine described below **exec-gate** performs state transition.

**exec-net** (net) =
\[
\{ i \leftarrow 0; \\
env \leftarrow \text{exec-initialise-runtime-env}(\text{net}); \\
\text{exec-top-level-loop}(\text{net}, \text{env}, i) \}
\]

**exec-top-level-loop** (net, env, i) =
\[
i > \text{s-dim}(\text{net}) \rightarrow \{ \text{PASS} \leftarrow \text{null} \}
\]

\[
T \rightarrow \{ \text{machine} \leftarrow \text{exec-get-code}(i, \text{net}); \\
i \leftarrow \text{exec-gate}(\text{net}, \text{env}, \text{machine}, i); \\
\text{exec-top-level-loop}(\text{net}, \text{env}, i) \}
\]

**exec-gate** (net, env, machine, i) =
\[
is\text{-interface-node}(\text{machine}) \rightarrow \{ \text{PASS} \leftarrow i + 1 \}
\]

\[
is\text{-noop}(\text{machine}) \rightarrow \{ \text{PASS} \leftarrow i + 1 \}
\]

\[
is\text{-constant}(\text{machine}) \rightarrow \\
\{ \text{exec-set-value}(\text{env}, i, \text{s-input}(\text{machine})); \\
\text{PASS} \leftarrow i + 1 \}
\]

\[
is\text{-latch}(\text{machine}) \rightarrow \\
\{ \text{previous} \leftarrow \text{exec-get-latch}(\text{env}, \text{s-latch-addr}(\text{machine})); \\
\text{current} \leftarrow \text{s-input}(\text{machine}); \\
\text{exec-set-value}(\text{env}, i, \text{previous}); \\
\text{exec-set-latch}(\text{env}, \text{s-latch-addr}(\text{machine}), \text{current}) \}
\]

\[
is\text{-ifm}(\text{machine}) \rightarrow \{ \text{exec-ifm-test}(\text{env}, \text{machine}, i) \}
\]

\[
is\text{-jump}(\text{machine}) \rightarrow \{ \text{s-input}(\text{machine}) \}
\]

\[
is\text{-set-node}!(\text{machine}) \rightarrow \\
\{ \text{exec-set-result}(\text{env}, \text{s-result}(\text{machine})), \\
\text{exec-get-value}(\text{env}, \text{s-input}(\text{machine})); \\
\text{PASS} \leftarrow i + 1 \}
\]

\[
is\text{-set-wire}!(\text{machine}) \rightarrow \\
\{ \text{exec-set-value}(\text{env}, \text{s-output}(\text{machine})), \\
\text{exec-get-value}(\text{env}, \text{s-input}(\text{machine})); \\
\text{PASS} \leftarrow i + 1 \}
\]

\[
is\text{-eqm}(\text{machine}) \rightarrow \{ \text{exec-eqm-test}(\text{env}, \text{machine}, i) \}
\]
is-gtm(machine) → \{ exec-gtm-test(env, machine, i) \}

is-ltm(machine) → \{ exec-ltm-test(env, machine, i) \}

is-saym(machine) → \{ write(s-input(machine)); PASS ← i + 1 \}

is-and(m) →
\{ exec-set-value(env, i, ∨\cap_{j=1}^n exec-get-value(env, s-input_j(machine))); PASS ← i + 1 \}

is-or(m) →
\{ exec-set-value(env, i, ∨\cap_{j=1}^n exec-get-value(env, s-input_j(machine))); PASS ← i + 1 \}

is-invert(machine) →
\{ exec-set-value(env, i, ¬ exec-get-value(env, s-input(machine))); PASS ← i + 1 \}

We have now described the main part of the abstract machine, which has a top-level-loop that gets the gate instruction at the current address, i, and passes this to exec-gate to obtain the next state address. As a side-effect this process alters the values held in the runtime environment env.

Some of the low level definitions are as follows.

The abstract machine uses a number of lower level definitions which can be collected together into four groups, as follows. The first group is the representation of the boolean values used at runtime, there are a number of options including using the symbols TRUE and FALSE as given here, or the numerals 1 and 0, which may lead to a less complex implementation.

**exec-set-true** (env, i) =
\{ exec-set-value(env, i, TRUE); PASS ← i + 1 \}

**exec-set-false** (env, i) =
\{ exec-set-value(env, i, FALSE); PASS ← i + 1 \}

atomic-to-bool (B) =
\begin{align*}
B = \text{TRUE} & \rightarrow \{ \text{PASS} ← T \} \\
B = \text{FALSE} & \rightarrow \{ \text{PASS} ← \text{null} \}
\end{align*}

T → error

The boolean values are used by the following boolean tests.

**exec-ifm-test** (env, machine, i) =
atomic-to-bool(exec-get-value(env, s-input_1(machine))) → \{ PASS ← i+1 \}

T → \{ PASS ← s-input_2(machine) \}

**exec-eqm-test** (env, machine, i) =
exec-get-value(env, s-input_1(machine)) =
exec-get-value(env, s-input_2(machine)) → \{ exec-set-true(env, i) \}

T → \{ exec-set-false(env, i) \}

**exec-gtm-test** (env, machine, i) =
exec-get-value($env$, $s\text{-input}_1$(machine)) >
exec-get-value($env$, $s\text{-input}_2$(machine)) \rightarrow \{ \text{exec-set-true}($env$, $i$) \}

$T \rightarrow \{ \text{exec-set-false}($env$, $i$) \}$

exec-ltm-test ($env$, machine, $i$) =
exec-get-value($env$, $s\text{-input}_1$(machine)) <
exec-get-value($env$, $s\text{-input}_2$(machine)) \rightarrow \{ \text{exec-set-true}($env$, $i$) \}

$T \rightarrow \{ \text{exec-set-false}($env$, $i$) \}$

The net formed during network analysis is used in two ways at runtime. exec-get-code is used by exec-top-level-loop to select the gate description at the current address, $i$, from net. The path used is also shown in figure E.5. The net is also used by exec-initialise-runtime-env

\[\text{exec-get-code} (i, \text{net}) =\]
\[\{ <\text{s-gate} \circ \text{s-addr}_i \circ \text{s-code} : \text{net}> \}\]

\[\text{exec-initialise-runtime-env} (\text{net}) =\]
\[\{ \text{env} \leftarrow \mu(\Omega ; <\text{s-latch-array} : \text{make-entry-store}(\text{s-dim} \circ \text{latch}(\text{net})))>,\]
\[<\text{s-result-array} : \text{make-entry-store}(\text{s-dim} \circ \text{s-result}(\text{net})))> ,\]
\[<\text{s-value-array} : \text{make-entry-store}(\text{s-dim}(\text{net})))> ;\]
\[\text{PASS} \leftarrow \text{env} \}\]

Getting and setting values.

\[\text{exec-get-value} (\text{env}, \text{addr}) =\]
\[\{ \text{PASS} \leftarrow \text{s-entry}_{\text{addr}} \circ \text{s-value-array} (\text{env}) \}\]

\[\text{exec-set-value} (\text{env}, \text{addr}, \text{val}) =\]
\[\{ \mu(\text{env}; <\text{s-entry}_{\text{addr}} \circ \text{s-value-array} : \text{val}>) \}\]

\[\text{exec-get-latch} (\text{env}, \text{addr}) =\]
\[\{ \text{PASS} \leftarrow \text{s-entry}_{\text{addr}} \circ \text{s-latch-array} (\text{env}) \}\]

\[\text{exec-set-latch} (\text{env}, \text{addr}, \text{val}) =\]
\[\{ \mu(\text{env}; <\text{s-entry}_{\text{addr}} \circ \text{s-latch-array} : \text{val}>) \}\]

\[\text{exec-set-result} (\text{env}, \text{addr}, \text{val}) =\]
\[\{ \mu(\text{env}; <\text{s-entry}_{\text{addr}} \circ \text{s-result-array} : \text{val}>) \}\]

This completes our description of the operational semantics for the gate language. Here we have discussed the BNF syntax of the various gates, an abstract syntax that can be used to represent the gates, a process called network analysis to convert a circuit-tree of gates into a linear structure, and finally a technique for runtime execution.
3 MACNET

In this section we will describe how the gate language can be enriched to allow the expression of rules. These rules could be written directly using the gate language however, this task is likely to be complex due to the number of gates involved. The MACNET language is designed to make this task easier by providing functionality that replaces repeatedly used collections of gates. The rules form a network called a MACNET. The MACNET language used to write the rules in a MACNET are based upon Agre’s “RUNNING ARGUMENTS” [2] and Chapman’s description of it [34]. Agre describes how the RUNNING ARGUMENTS involve putting forward proposals and objections that can support and override one another reflecting the dynamics of the knowledge they represent. The RUNNING ARGUMENT rules themselves take two forms, if rules and unless rules, that correspond to combinatorial logic with an if being an AND and unless being a NOT-AND. Figure E.6 depicts two example rules which in addition to providing examples of the if and unless form also show two different ways in which the argument structure of the RUNNING ARGUMENT rules affects the circuit created.

The MACNET language does not use the RUNNING ARGUMENT’s rule form instead it uses arbiters. The rules expressed in MACNET form the central system, which operates in conjunction with a set of operators in the peripheral system that are able to perform actions and obtain new values. This top-level-loop does one iteration of

1. run peripheral system
2. run central system

each clock tick. At any one time a subset of selected operators are run providing input values for the MACNET that it uses during run central system. When the MACNET is run it provides output that selects which operators should be run on what values for the next clock tick. Figure E.7 shows the interface between operators and MACNET, showing the ready and enable flags that indicate when an operator result or MACNET arbiter result is available, respectively. Note that an arbiter’s result is the argument input to an effector, which in most cases is an operator that has the same name as the arbiter.

Arbiters are similar to a combined form of the if and unless rules although not as intuitively simple. The if rule in figure E.6 captures an essential arbiter property concerning how proposed wire values are selected. In figure E.6 we have the choice from three values of foo (a, b or c) that are to be passed onto the wire bar. The arbitration language provides a declarative way of stating how an “agent” in the world may react when given a certain input. The values given to an agent do not need to be boolean, we can also use integers and symbols. Inside the network there are two types of wire: wires that hold input signals, and wires that hold the internally generated boolean values. The circuit language primitives that primarily operate on the input signals are gates like gtm and eqm, that can compare boolean and non-boolean (e.g., integer) values producing a boolean result that can be used by the other gates (e.g., andg, etc). These primitives provide some flexibility in the form of the declarations made.
Although the MACNET uses a simple value set, the operators that it works with can use a more complicated model of the world. The MACNET language has four main components: registrars, arbiters, conditions and proposers. In addition to these main components are operator and pkg (which is short for package) components. Operators are really outside a MACNET being those functions that a MACNET is designed to select, and who also provide the input a MACNET is to run upon. Some operators are parameterised by a special doit? flag that makes the operator concerned a no-op unless the flag is high.

PQks on the other hand are used as a structure within which to construct a MACNET. A pkg allows MACNET components to be added incrementally, to build a single MACNET identified by its pkg name P. The three main pkg language functions are: (make-pkg) which creates and returns an empty pkg structure, (with-pkg P &rest body) which opens pkg P allowing MACNET language declarations to be added in the with-pkg body. (compile-pkg P) which compiles the arbiter language definition held in P into an executable form which it returns.
3.1 Language components

Registrars Registrars are used to declare any simple circuit descriptions that produce a result which is to be used by more than one arbiter, providing a useful pre-processing stage. These functions have the form (registrar register-name wire). The input values from the outside world are generally fed through registrars, performing any initial calculations and making rules written in the MACNET language clearer by allowing the use of each register-name instead of the set of gates it represents.

Arbiters Arbiters are important elements in this language because they facilitate the selection of which set of values are to be assigned to a given set of wires. There are two forms of arbitration: (1) the abstract-arbiter which allows internal value selections to be made that can used in further arbitration, and (2) the arbiter which selects which set of values should be given to the operator that it represents. Figures E.8 and E.9 show the ordering between abstract-arbiters and arbiters. The abstract form (abstract-arbiter name arglist &rest body) produces a result that can be used inside the pkg it is defined in. The argument list of the abstract arbiter defines the names of the arbiter's ports which can be accessed by using (port abstract-operation-name port-name). As shown in figure E.10, port returns the bus that is the port named port-name of the abstract operation named abstract-operation-name.

An arbiter (arbiter name arglist &rest body) produces a result that is exported as the result for the operator, name, which is given the selected values for arglist. Once either arbiter form has been created it can later be added to, at a later date, by using the form (with-arbiter name &rest body). All these arbiters have the same form and we describe next how they allow proposals to be put forward and overridden.

Proposals Each proposer has an identifying name and gives the result variables a proposed value or (more correctly) a circuit that will evaluate to a value. A proposal is the set of actual parameters that a proposer thinks the named operator should be given at a particular time. For example: (propose marker-behind :marker *nearest-marker* :testfor (constant :overlap) :doit? (constant *tt*)) A proposal can take one of two forms, a default (propose-default proposer-name &rest key-arglist) or a general proposal (propose proposer-name &rest key-arglist) which can be supported by the use of one or more conditions. The difference between these two forms of proposal lies in their precedence. Propose-default has zero precedence and is overridden by any other proposal. If more than one propose is satisfied at any one time, the proposal with the highest precedence is selected. This precedence ordering is defined by a declaration using the function (override-proposer overriding-proposer &rest overridden-proposers) which ensures that overriding proposal has a higher precedence than the other named overridden-proposers. override-proposer ensures that the named proposer is preferred to its overridden-proposers. The ability to override is determined by assigning precedence
to the proposals that interact. Sometimes proposals are mutually exclusive, so no additional ordering is required.

To ensure that one proposer is valid for an arbiter at any one time, runtime error checking is performed. This error checking takes two forms: (1) that not more than one proposer is satisfied, and (2) that no proposer is satisfied. The first case can be solved by using `override-proposer` to better define the precedence order or by using (indifferent & rest proposers) which removes error checking for the case and instead arbitrarily chooses one of the satisfied proposers.

Conditions The condition is a separate clause stating the name of its proposal that it is a condition of and the circuit that is to be evaluated. Proposals are satisfied by their conditions being met. Conditions are defined separately from a propose function by using (condition proposer-name wire). A condition specifies the situation under which the named proposer is satisfied, e.g., (condition marker-behind (andg
If more than one condition is present they act conjunctively; if any of them are false the proposer does not propose anything.

If a proposal is not given a condition it may never be fired so a compile time error is given; this is different for the default-proposal's which are always true and do not need a condition or arbiters that only have one proposed.

3.2 Extensions

The gate language can be easily extended. For example as part of the MACNET language a transm gate was added, which was intended to provide a boolean result characterising the likelihood of a transition from one integer value to another (i.e., for the previous and current values of some wire). This gate was not described before since it uses a MACNET primitive to set the "matrix of values" that is to be used by a given transm at execution time.

A transm takes three inputs. The first two are wires representing values for previous and current. The third value is a matrix name that has been defined by the MACNET function (trans-matrix name contents). The BNF syntax needs to be extended to include transm:

```
<gate machine> ::= 
                  |
[Previous <gate machine> definitions]
                  |
                  |
| transm [ <wire>; <wire>; <atomic> ]
```

The abstract syntax requires one rewrite of the is-ternary-gate definition to reflect the addition of the new ternary gate transm.

```
is-ternary-gate = ( <s-input_1 : is-gate>, <s-input_2 : is-gate>,
                  <s-input_3 : is-gate>, <s-gate-name : is-ifm V is-transm> )
```

The instruction definition for the abstract machine requires of two additions: one for network analysis and the other for runtime execution. In network analysis we extend network-analysis-ternary-gate definition as follows:

```
network-analysis-ternary-gate (net, machine) =
                  |
[Previous network-analysis-ternary-gate definitions]
                  |
                  |
| is-transm(machine) →
{ address_1 ⇐ network-analysis(net, s-input_1(machine));
  address_2 ⇐ network-analysis(net, s-input_2(machine));
  matrix ⇐ s-input_3(machine);
  P= ( <s-input_1 : address_1 >, <s-input_2 : address_2 >,
       <s-input_3 : matrix >);
store(\text{net}, \text{s-gate-name}(\text{machine}), \mathcal{P}). \} \\

In the instruction definition for runtime execution we add a new clause to \texttt{exec-gate} as follows:

\begin{verbatim}
exec-gate (\text{net}, \text{env}, \text{machine}, i) = \\
    \text{:} \\
    \text{[Previous exec-gate definitions]} \\
    \text{:} \\
    \text{is-trans}(\text{machine}) \rightarrow \\
    \{ M \leftarrow \text{get-transition-matrix}(\text{env}, \text{exec-get-value}(\text{env}, s-\text{input}_a(\text{machine}))); \text{exec-set-value}(\text{env}, i, M[s-\text{input}_1(\text{machine})][s-\text{input}_2(\text{machine})]); \text{PASS} \leftarrow i + 1 \} \\
\end{verbatim}

This completes our description of the \texttt{trans} extension to the gate language.
4 MACNET LANGUAGE DETAILS

We have given an informal description of the MACNET language, parts of which we will present more formally. The mapping MACNET language to GATE language is given here in three stages corresponding the functions form-equiv, cons-pkg and compile-pkg, which are related as follows

\[ \text{MACNET-} \beta \xrightarrow{\text{form-equiv}} \text{MACNET-} \alpha \xrightarrow{\text{cons-pkg}} \text{MACNET-} \sigma \xrightarrow{\text{compile-pkg}} \mathcal{G} \]

In stage one we present the BNF syntax of MACNET-\(\beta\) together with the abstract syntax, MACNET-\(\alpha\) and discuss the equivalence relationship that hold between the two forms. We do not describe the function form-equiv which could be performed as part of a macro-expansion. In stage two the instruction definitions of the MACNET-\(\alpha\) forms are given, which construct the is-pkg structure the MACNET-\(\sigma\) form. Once we have the MACNET-\(\sigma\) form it can be translated into gate language via compile-pkg : MACNET-\(\sigma\) \(\rightarrow\) \(\mathcal{G}\).

To begin with let us describe the syntax of the MACNET language to be discussed here. We use the same BNF notation as in section 2.

\[
\begin{align*}
<\text{macnet pkg}> &::= \quad \text{with-pkg} [ <\text{name}>; <\text{macnet}> ] \\
<\text{macnet}> &::= \quad <\text{registrar}> | <\text{arbitration}> | <\text{trans matrix}> \\
<\text{registrar}> &::= \quad \text{registrar} [ <\text{name}>; <\text{wire}> ] \\
<\text{trans matrix}> &::= \quad \text{trans-matrix} [ <\text{name}>; <\text{matrix}> ] \\
<\text{arbitration}> &::= \quad \text{arbiter} [ <\text{name}>; <\text{key list}>; <\text{arbiter body}> ] \\
& \quad | \text{abstract-arbiter} [ <\text{name}>; <\text{key list}>; <\text{arbiter body}> ] \\
& \quad | \text{with-arbiter} [ <\text{name}>; <\text{arbiter body}> ] \\
<\text{arbiter body}> &::= \quad <\text{component}>; \ldots; <\text{component}> \\
<\text{component}> &::= \quad <\text{proposer}> | <\text{condition}> | <\text{override}> | <\text{indifferent}> \\
<\text{proposer}> &::= \quad \text{propose-default} [ <\text{name}>; <\text{key arg}>; \ldots; <\text{key arg}> ] \\
& \quad | \text{propose} [ <\text{name}>; <\text{key arg}>; \ldots; <\text{key arg}> ] \\
<\text{condition}> &::= \quad \text{condition} [ <\text{name}>; <\text{wire}> ] \\
<\text{override}> &::= \quad \text{override-proposer} [ <\text{name}>; <\text{name}>; \ldots; <\text{name}> ] \\
<\text{indifferent}> &::= \quad \text{indifferent} [ <\text{name}>; \ldots; <\text{name}> ] \\
<\text{key arg}> &::= \quad <\text{keyword}>; <\text{argument}> \\
<\text{port}> &::= \quad \text{port} [ <\text{name}>; <\text{keyword}> ] \\
<\text{key list}> &::= \quad [ <\text{Lisp symbol}>; \ldots; <\text{Lisp symbol}> ] \\
<\text{name}> &::= \quad <\text{Lisp symbol}> \\
<\text{keyword}> &::= \quad <\text{Lisp keyword}> \\
<\text{argument}> &::= \quad <\text{wire}> \\
<\text{matrix}> &::= \quad [ <\text{row}>; \ldots; <\text{row}> ] \\
<\text{row}> &::= \quad [ <\text{integer}>; \ldots; <\text{integer}> ]
\end{align*}
\]

Redefine:

\[
<\text{wire}> ::= \quad <\text{gate machine}> | <\text{port}>
\]
We have redefined the definition of \(<\text{wire}>\) to allow for \text{ports} to be included in the gate language forms and be part of the MACNET language made visible to a person using the language. The compile-pkg process requires the following abstract syntax, defining the is-pkg, is-arbiter and is-propose representations. The BNF syntax form of the MACNET language is mapped onto an abstract machine that uses the following abstract syntax:

\[
\begin{align*}
\text{is-pkg} &= ( \text{<s-reg-names>: is-list >}, \text{<s-reg-table>: is-entrystore >}, \\
& \quad \text{<s-arg-list>: is-entrystore >}, \text{<s-op-names>: is-list >}) \\
\text{is-arbiter} &= ( \text{<s-output: is-list >}, \text{<s-indiff-props: is-entrystore >}, \\
& \quad \text{<s-indiff-rels: is-entrystore >}, \text{<s-prop-table: is-entrystore >}, \\
& \quad \text{<s-arg-type: is-atomic >}) \\
\text{is-propose} &= ( \text{<s-args: is-list >}, \text{<s-conditions: is-list >}, \\
& \quad \text{<s-overridden-by: is-list >})
\end{align*}
\]

The \text{with-pkg} form, described earlier, allows all related definitions to be composed into one is-pkg structure, that can then be compiled. Note that (\text{with-pkg \text{P \&rest body}}) puts \text{P} the package name in scope of all MACNET language declarations made in \text{body}. Given this we can make the following equivalence relations:

\[
\begin{align*}
\text{registrar name wire} & \equiv \{ \text{macnet-registrar(\text{P, name, wire})} \} \\
\text{arbiter name arglist \&rest body} & \equiv \{ \text{ARBNNAME} \Leftarrow \text{name;}
\quad \text{macnet-arbiter(\text{P, name, arglist});} \\
\quad \text{macnet-arb-body(\text{P, ARBNNAME, body})} \} \\
\text{abstract-arbiter name arglist \&rest body} & \equiv \{ \text{ARBNNAME} \Leftarrow \text{name;}
\quad \text{macnet-abstract-arbiter(\text{P, name, arglist, body});} \\
\quad \text{macnet-arb-body(\text{P, ARBNNAME, body})} \}
\end{align*}
\]

\[
\text{macnet-arb-body (P, ARBNNAME, body)} = \\
\text{is\emptyset(body) } \rightarrow \text{null}
\]

\[
\text{T } \rightarrow \{ \text{macnet-component(\text{P, ARBNNAME, head(body));}} \\
\quad \text{macnet-arb-body(\text{P, ARBNNAME, tail(body)})} \}
\]

\text{macnet-component} is more difficult to provide an instruction definition for, since it uses further equivalence relationships that we will consider next. Note that in addition to \text{P}, the variable \text{ARBNNAME} is now in scope within the \text{body} of an arbiter where the following can be declared:

\[
\text{propose-default name} \&rest \text{key-arglist} \equiv \{ \text{macnet-propose-default(\text{P, ARBNNAME, name, key-arglist})} \}
\]
(propose name &rest key-arglist) \equiv \{ \text{macnet-propose}(P, \text{ARBNAME}, \text{name}, \text{key-arglist}) \} \\
(condition name wire) \equiv \{ \text{macnet-condition}(P, \text{ARBNAME}, \text{name}, \text{wire}) \} \\
(override-proposer overriding-proposer &rest overridden-proposers) \equiv \{ \text{macnet-override-proposer}(P, \text{ARBNAME}, \text{overriding-proposer}, \text{overridden-proposers}) \} \\
(indifferent &rest proposers) \equiv \{ \text{macnet-indifferent}(P, \text{ARBNAME}, \text{proposers}) \} \\

Having defined the equivalence mapping between the BNF syntax that defines cons-pkg and abstract syntax let us now describe the instruction definition of each MACNET function.

\text{macnet-registrar } (P, \text{name}, \text{wire}) = \\
\{ \mu(P) < \text{s-reg-names} \circ \text{s-pkg} : \text{name} \mapsto \text{s-reg-names} \circ \text{s-pkg}(P) >); \\
\mu(P) < \text{s-entry}_\text{name} \circ \text{s-reg-table} \circ \text{s-pkg}(P) > \} \\

\text{macnet-arbiter } (P, \text{name}, \text{arglist}) = \\
\{ \mu(P) < \text{s-op-names} \circ \text{s-pkg} : \text{name} \mapsto \text{s-op-names} \circ \text{s-pkg}(P) >); \\
S \leftarrow \text{s-entry}_\text{name} \circ \text{s-arb-table} \circ \text{s-pkg}(P); \\
\mu(P) < \text{s-output} \circ S(P) : \text{arglist} >, \\
< \text{s-indiff-vars} \circ S(P) : \text{make-entry-store}(\text{maxlen}) >, \\
< \text{s-indiff-vars} \circ S(P) : \text{make-entry-store}(\text{maxlen}) >, \\
< \text{s-prop-table} \circ S(P) : \text{make-entry-store}(\text{maxlen}) >, \\
< \text{s-arb-type} \circ S(P) : \text{arbiter} > \} \\

\text{macnet-abstract-arbiter } (P, \text{name}, \text{arglist}) = \\
\{ S \leftarrow \text{s-entry}_\text{name} \circ \text{s-arb-table} \circ \text{s-pkg}(P); \\
\mu(P) < \text{s-output} \circ S(P) : \text{arglist} >, \\
< \text{s-indiff-vars} \circ S(P) : \text{make-entry-store} >, \\
< \text{s-indiff-vars} \circ S(P) : \text{make-entry-store} >, \\
< \text{s-prop-table} \circ S(P) : \text{make-entry-store} >, \\
< \text{s-arb-type} \circ S(P) : \text{abstract} > \} \\

Arbiters are typed by the use of a keyword that is either arbiter or abstract. We will describe the possible components of an arbiter's body.

\text{macnet-propose-default } (P, \text{ARBNAME}, \text{name}, \text{key-arglist}) = \\
\{ S_{arb} \leftarrow \text{s-entry}_{\text{ARBNAME}} \circ \text{s-arb-table} \circ \text{s-pkg} ; \\
\mu(P) < \text{s-args} \circ \text{s-entry}_\text{name} \circ S_{arb} : \text{key-arglist} > \\
\text{macnet-overrideloop}(P, S_{arb}, \text{everything}, \text{name} \mapsto \emptyset) \} \\

The keyword everything is replaced during compile-pkg by a list of all the other proposers for \text{ARBNAME}, so that this proposer is the default action for \text{ARBNAME}.  

\textbf{macnet-propose} \((P, \text{ARBNAME}, \text{name}, \text{key-arglist}) = \)
\begin{align*}
\{ & S_{arb} \leftarrow s\text{-entry}_{\text{ARBNAME}} \circ s\text{-arb-table} \circ s\text{-pkg} ; \\
& \mu(P ; s\text{-args} \circ s\text{-entry}_{\text{name}} \circ S_{arb} : \text{key-arglist}) \} \\
\end{align*}

\textbf{macnet-condition} \((P, \text{ARBNAME}, \text{name}, \text{wire}) = \)
\begin{align*}
\{ & S_{prop} \leftarrow s\text{-entry}_{\text{name}} \circ s\text{-entry}_{\text{ARBNAME}} \circ s\text{-arb-table} \circ s\text{-pkg} ; \\
& \mu(P ; <s\text{-conditions} \circ S_{prop} : \text{wire} \uplus s\text{-conditions} \circ S_{prop}(P)) >) \} \\
\end{align*}

\textbf{macnet-override-proposer} \((P, \text{ARBNAME}, \text{overriding-proposer}, \text{overridden-proposers}) = \)
\begin{align*}
\{ & S_{arb} \leftarrow s\text{-entry}_{\text{ARBNAME}} \circ s\text{-arb-table} \circ s\text{-pkg} ; \\
& \text{macnet-overrideloop}(P, S_{arb}, \text{overriding-proposer}, \text{overridden-proposers}) \} \\
\end{align*}

\textbf{macnet-override-loop} \((P, S_{arb}, \text{overriding-proposer}, oplist) = \)
\begin{align*}
is\emptyset(oplist) & \rightarrow \text{null} \\
T & \rightarrow \{ \text{name} \leftarrow \text{head}(oplist); \\
& S_{prop} \leftarrow s\text{-entry}_{\text{name}} \circ S_{arb}; \\
& \mu(P ; <s\text{-overridden-by} \circ S_{prop} : \text{overriding-proposer} \uplus \\
& \text{s-overridden-by} \circ S_{prop}(P)) >); \\
& \text{macnet-override-loop}(P, S_{arb}, \text{overriding-proposer}, \text{tail}(oplist)) \} \\
\end{align*}

\textbf{macnet-indifferent} \((P, \text{ARBNAME}, \text{proposers}) = \)
\begin{align*}
\{ & S_{arb} \leftarrow s\text{-entry}_{\text{ARBNAME}} \circ s\text{-arb-table} \circ s\text{-pkg} ; \\
& D \leftarrow s\text{-dim} \circ s\text{-indiff-rels} \circ S_{arb}(P); \\
& \mu(P ; <s\text{-dim} \circ s\text{-indiff-rels} \circ S_{arb} : (1 + D)) >); \\
& \mu(P ; <s\text{-entry}_{D} \circ s\text{-indiff-rels} \circ S_{arb} : \text{proposers}) >); \\
& \text{macnet-indiff-loop}(P, S_{arb}, D, \text{proposers}) ; \\
\end{align*}

\textbf{macnet-indiff-loop} \((P, S_{arb}, D, proplist) = \)
\begin{align*}
is\emptyset(proplist) & \rightarrow \text{null} \\
T & \rightarrow \{ \text{name} \leftarrow \text{head}(proplist); \\
& \mu(P ; <s\text{-entry}_{\text{name}} \circ s\text{-indiff-props} \circ S_{arb} : D \uplus \\
& s\text{-entry}_{\text{name}} \circ s\text{-indiff-props} \circ S_{arb}(P)) >); \\
& \text{macnet-indiff-loop}(P, S_{arb}, D, \text{tail}(proplist)) \} \\
\end{align*}

This has described the instruction definitions for \texttt{cons-pkg}, which converts the MACNET-\(\alpha\) form to a MACNET-\(\sigma\) form that can be mapped onto the gate language using \texttt{compile-pkg}. Figures E.8 and E.9 show the ordering amongst the MACNET language components, which needs to be present in the gate language version. During the \texttt{compile-pkg} translation, interrelationships between different language components and within a language component are resolved by using a topological sort to determine all dependencies. Any loops detected are raised as errors.

Some parts of the MACNET language do not require much transformation because they are already written using gate language, for example the values to the arglists for proposers and conditions. However, ports need to be identified, as they represent input from an abstract arbiter and each port is replaced by an \texttt{interface-wire}.

The MACNET components form an acyclic graph with directed edges from inputs to outputs. The graph can be treated as being composed from subgraphs representing
registrars and arbiters, where each arbiter subgraph is made up of its proposer graph with directed edges from each proposer to those it overrides. This graph structure allows us to use a topological sort to order the graph nodes so that in our gate-language version gate-input-values are always available before they are used. The general stages of the translation algorithm, compile-pkg, used to construct the gate language version, GC, are:

1. \( GC \leftarrow \emptyset \)

2. Order the dependencies between arbiters based on arbiter input from abstract arbiters.

3. For each arbiter
   3.i. Order the proposals the arbiter uses based upon overriding proposers, so that all overriding proposers are placed before those that they override.
   3.ii. \( GC \leftarrow (\text{build-arbiter-proposer}) \# GC \)

4. Order the registrars.

5. For each registrar
   5.i. \( GC \leftarrow (\text{build-registrar}) \# GC \)

6. Return "sequ" \( \# GC \)

Here we have introduced the use of "__" where the pair of double quotes encloses the underline which represents any valid symbol or sequence of symbols from the gate language. This is to shield the quoted symbol or symbols from immediate evaluation and is the output from this translation process. We also use \( \# \) to compose these quoted output structures.

The output from compile-pkg is a textual representation of the GC, which is then further compiled as described in section 2 by using do-network-analysis. Figure E.11 illustrates the key parts of the compile-pkg algorithm. In the description here the textual form follows the gates BNF syntax, an alternative to this would be to use the equivalent abstract syntax, however this is an internal representation and implementation dependent making it less attractive as an intermediate form.

Let us now consider the semantics of build-arbiter-proposer:
build-arbitr-proposer \((P, ARBNAME, args, proposers, indifferent) = \)
\[
\{ \text{propm} = \text{build-proposer-machine}(P, ARBNAME, proposers) ; \\
\text{PASS} \Leftarrow \text{build-arb-test}(P, ARBNAME, args, proposers, indifferent, propm) \}
\]

build-arb-test \((P, ARBNAME, args, proposers, indifferent, propm) = \)
\[
is\emptyset(propm) \rightarrow \\
\{ \text{PASS} \Leftarrow \text{build-arb-single}(ARBNAME, args, head(proposers)) \}
\]

\(T \rightarrow \{ \text{PASS} \Leftarrow \text{build-arb-multi}(ARBNAME, args, proposers) \}
\]

build-arb-multi \((P, ARBNAME, args, proposers, indifferent) = \)
\[
\{ \text{testwires} \Leftarrow \text{build-arb-ambiguity}(ARBNAME, proposers, indifferent) ; \\
\text{outputwires} \Leftarrow \text{build-arb-prop-type}(P, ARBNAME, args, proposers, indifferent) ; \\
\text{PASS} \Leftarrow \text{sequ} \oplus \text{propm} \oplus \text{testwires} \oplus \text{outputwires} \}
\]

build-arb-ambiguity \((ARBNAME, proposers, indifferent) = \)
\[
\{ \ldots \}
\]

Figure E.12 shows that the test for correct evaluation of an arbiter has two parts. In the
figure, we have a set of three input wires \(a, b\) and \(c\), and the output wires \(w\) and \(e\). An
org is used to determine if the arbiter has any inputs this gives a value to \(w\). If \(w\) is false
then give a warning that there has been no call to this operation. A pairing of all inputs
are sent to an org to check whether two or more are true at once, giving the wire \(e\) a
value. If \(e\) is true then there has been an ambiguous call to the arbiter.

Note that an "indifferent" declaration between input wires is equivalent to an org that
replaces the inputs that are declared indifferent by the org's result.

build-arb-prop-type \((P, ARBNAME, args, proposers, indifferent) = \)
\[
s\text{-arb-type} \circ s\text{-entry}_{ARBNAME} \circ s\text{-arb-table} \circ s\text{-pkg}(P) = \text{arbiter} \rightarrow \\
\{ \text{PASS} \Leftarrow \text{build-multi-arbiter} \}
\]

\[s\text{-arb-type} \circ s\text{-entry}_{ARBNAME} \circ s\text{-arb-table} \circ s\text{-pkg}(P) = \text{abstract} \rightarrow \\
\{ \text{PASS} \Leftarrow \text{build-multi-abstract} \}
\]

cons-name \((lis) = \)
\[
\{ str \Leftarrow "n" ; \\
\text{PASS} \Leftarrow \text{cons-name-loop}(lis, str) \}
\]
cons-name-loop \( (\text{lis}, \text{str}) = \)
\[
\begin{align*}
is\emptyset(\text{lis}) & \rightarrow \{ \text{PASS} \leftarrow \text{str} \} \\
T & \rightarrow \{ \\
& \quad \text{str} \leftarrow \text{"\text{-}" \@ head(\text{lis});} \\
& \quad \text{PASS} \leftarrow \text{cons-name-loop(head(\text{lis}), str)} \}
\end{align*}
\]

cons-args \( (\text{ARBNAME}, \text{arglist}, \text{prop}, \text{GC}) = \)
\[
\begin{align*}
is\emptyset & \rightarrow \{ \text{PASS} \leftarrow \text{GC} \} \\
T & \rightarrow \{ \\
& \quad \text{a} \leftarrow \text{head(\text{arglist});} \\
& \quad \text{output-name} \leftarrow \text{cons-name}(\text{ARBNAME}++\text{a}++\emptyset); \\
& \quad \text{prop-arg-name} \leftarrow \text{cons-name}(\text{ARBNAME}++\text{prop}++\text{a}++\emptyset); \\
& \quad \text{node} \leftarrow \text{"set-node!" ++ output-name ++ prop-arg-name;} \\
& \quad \text{GC} \leftarrow \text{node ++ GC;} \\
& \quad \text{cons-args(ARBNAME, tail(\text{arglist}), \text{prop}, \text{GC}) } \}
\end{align*}
\]

cons-props \( (\text{ARBNAME}, \text{arglist}, \text{proplist}, \text{GC}) = \)
\[
\begin{align*}
is\emptyset & \rightarrow \{ \text{PASS} \leftarrow \text{GC} \} \\
T & \rightarrow \{ \\
& \quad \text{prop} \leftarrow \text{head(\text{proplist})} \\
& \quad \text{test} \leftarrow \text{"interface-wire!" ++ prop ++ \emptyset;} \\
& \quad \text{setfun} \leftarrow \text{cons-args(ARBNAME, arglist, prop, \emptyset);} \\
& \quad \text{item} \leftarrow \text{test ++ setfun;} \\
& \quad \text{GC} \leftarrow \text{item ++ GC;} \\
& \quad \text{cons-props(ARBNAME, arglist, tail(\text{proplist}), \text{GC}) } \}
\end{align*}
\]

The definition of build-multi-arbiter is constructing a \texttt{condm} gate structure that has the following form:

\[
\begin{align*}
\text{(condm} \ ( (\text{interface-wire!}...)) \\
& \quad (\text{set-node!}.......) \\
& \quad \vdots \\
& \quad (\text{set-node!}.......) \\
& \quad \text{ (otherwise (saym "error: no proposers for ..."))})
\end{align*}
\]

This structure is the same for \texttt{build-multi-abstract} except we use \texttt{set-wire!} instead of \texttt{set-node!}. The example in figure E.13 shows an arbiter that has three proposers with a \texttt{condm} represented by three of its ifm gates. The ifm boxes have switch on top that
enables the passing of the respective Arg values defined by each proposer key-args. The
values given to the switches depend upon the condition wires from input and the overridden
status. The proposer precedence here is represented by the andg gates P1, P2, P3.

\[
\text{build-multi-arbiter (ARBNAME, arglist, proplist) =}
\{ \text{PASS \leftarrow "condm" \oplus cons-propsARBNAME, arglist, proplist, } \emptyset \}
\]

\[
\text{build-arb-single (ARBNAME, arglist, prop) =}
\{ \text{PASS \leftarrow cons-argsARBNAME, arglist, prop, } \emptyset \}
\]

The semantics for build-registrar are as follows:

\[
\text{build-registrar } P, name = \{ \text{PASS \leftarrow "set-wire!" \oplus name \oplus s-entry}_{name} \circ
s-reg-table \circ s-pkg(P) + \emptyset \}
\]

This completes the description of the MACNET language.
5 Summary

We have now described one version of the MACNET language. Chapman [34] provides details of possible approaches to debugging at runtime and also extra functionality that can be added to the MACNET language.

The gate language description in sections 1 and 2 explain how to make the static and runtime environments, and also how to execute them. After we have run the circuit, we want to run the selected operators. This is done via peripheral-execute, an iteration over the results stored in each arbiter's output nodes that calls the appropriate operator function on the arbiter output. This is made easier by arbiters having the same names as the operators that they represent. Once the operators have run their results act as input to the registrars, proposers and conditions, via the interface-node gates.

To make this description clearer, consider the example given in figures E.14 and E.15 which are taken from Chapman's description of BLOCKHEAD, a deictic system that operates in a blocks-world where it has to copy the right most stack. The code given here finds the right most stack. Chapman [33] describes this and how the rest of BLOCKHEAD operates. In the code sections given here, *copymarker* is being positioned so that it marks the block to be copied.
(defun blockhead-find-right-most-stack (Pkg) (with-pkg Pkg

(arbiter find-random-block! (marker doit?)
  (propose-default noop :marker (constant *f*) :doit? (constant *f*))
  (propose find-a-block :marker *copymarker* :doit? (constant *t*))
  (condition find-a-block (andg *find-right-most-stack* *boot*
    (invert *building-stack-copy*))))

(arbiter step-down-to-table! (marker doit?)
  (propose-default noop :marker (constant *f*) :doit? (constant *f*))
  (propose step-down :marker *copymarker* :doit? (constant *t*))
  (condition step-down (andg *find-right-most-stack*
    *find-random-block!-result*
    (invert *building-stack-copy*))))

(arbiter keep-off-table (marker doit?)
  (propose-default noop :marker (constant *f*) :doit? (constant *f*))
  (propose move-up-from-table :marker *copymarker* :doit? (constant *t*))
  (condition move-up-from-table (andg (invert *building-stack-copy*)
    *find-right-most-stack*
    (eqm *copymarker-on* (constant :table))))

(arbiter keep-finding-right-most-stack (doit?)
  (propose-default noop :doit? (constant *f*)
  (propose continue :doit? (constant *t*)
  (condition continue (andg (invert *boot*) (invert *building-stack-copy*)
    *find-right-most-stack*)))

(arbiter copymarker-celltype (marker doit?)
  (propose-default noop :marker (constant *f*) :doit? (constant *f*)
  (propose findout :marker *copymarker* :doit? (constant *t*)
  (condition findout (andg *find-right-most-stack* *boot*
    (invert (eqm *copymarker-on* (constant :table))))
  (propose newblock :marker *copymarker* :doit? (constant *t*)
  (condition newblock *building-stack-copy*
    (override-proposer findout newblock))

(arbiter goto-right-most-stack! (marker doit?)
  (propose-default noop :marker (constant *f*) :doit? (constant *f*)
  (propose find-stack :marker *copymarker* :doit? (constant *t*)
  (condition find-stack (andg *find-right-most-stack* *boot*
    (invert *building-stack-copy*)
    (eqm *copymarker-on* (constant :block))
    (invert *find-random-block!-result*)))))))

Figure E.14: An example from BLOCKHEAD: find-right-most-stack. These are the arbiters.
\begin{verbatim}
(defOperation find-random-block! (self doit?)
 (find-random-block bw markers self)
 (set-result! *find-right-most-stack* *t*)
 (set-result! *find-random-block!-result* *t*))

(defOperation step-down-to-table! (self doit?)
 ;; This should really consist of a set of routines that call lower-level
 ;; primitives to move-down the marker one step per clock cycle.
 ;; The operators could then be said to just use of a collection
 ;; of simple effectors.
 (move-to-bottom-of-stack stream bw markers self)
 (set-result! *find-right-most-stack* *t*)
 (set-result! *copymarker-on* :table)
 (set-result! *step-down-to-table!-result* *t*))

(defOperation keep-off-table (self doit?)
 ;; Move-up moves the marker self up one grid position.
 (move-up markers self))

(defOperation keep-finding-right-most-stack (doit?)
 (set-result! *find-right-most-stack* *t*))

(defOperation copymarker-celltype (self doit?)
 ;; Marker-cell-type-property returns the cell-type of the grid
 ;; it marks. The cell-types are :block, :table and :freespace.
 (set-result! *copymarker-on* (marker-cell-type-property bw markers self)))

(defOperation goto-right-most-stack! (self doit?)
 ;; Jump-right moves the marker self to next block on the right in the
 ;; same row, and if there is no next block it returns nil.
 ;; The loop is another simplification, we should really do
 ;; one jump per clock cycle.
 (loop (when (null (jump-right bw markers self))
 (return nil))
 (refresh-blocksworld stream bw markers))
 (set-result! *find-right-most-stack* *f*)
 (set-result! *building-stack-copy* *t*))
\end{verbatim}

Figure E.15: Continuing the \texttt{find-right-most-stack} example from BLOCKHEAD with
the operators. In the above: \texttt{bw} is the blocksworld data-structure, markers is the data-
structure that holds information about the markers, and \texttt{stream} is the display window.
IJCAI is an abbreviation for International Joint Conference on Artificial Intelligence. AAAI is an abbreviation for the National Conference on Artificial Intelligence, an annual conference of the American Association of Artificial Intelligence. ECAI is an abbreviation for European Conference on Artificial Intelligence. ECCV is an abbreviation for European Conference on Computer Vision.


[108] Richard J. Howarth and Andrew F. Toal. Qualitative space and time for vision. In The First Qualitative Vision Workshop, August 1990. Also as VIEWS project report MRC-03-FF-08.


BIBLIOGRAPHY


INDEX

2D+ t, 2D space and time, 35, 257

\( \square \), no current agent reference
\( O \)-notation, asymptotic upper bound, 63, 155
\( \Omega \)-notation, asymptotic lower bound, 63
\( \Theta \)-notation, worst case, 63, 155
accomplishment, 176, 244
active vision, 41, 49, 70, 209
\( \rightarrow \), add directed edge, 154
adjacency set, 56, 230

AGENCY
agents, 143–146
operators, 126–127
Agre, 37, 46, 112, 115, 123, 133, 137, 143, 175, 207

algorithms
continue, 159
focus, 159
in same region, 86
network expansion and inference, 150, 156
node-building, 164
node-linking, 166
positional feature, 263
terminate, 159
update THB, 98

ALLOCATE, 148
ALIVEN, 48, 253
\#’, list append, 286
arbiter, 144–146, 222

ART, the set of application domain region
types, 65

AS, the set of adjacency sets (cell descriptions), 65
aspect, 115, 124, 144
assumptions, 3–4, 209
attention, 42
early visual processing, 37
Ballard, 9, 40–41, 144, 209–211
Bayesian networks, 9, 45, 48, 101, 150–
161, 165, 266–271
BELIEVER, 44
BLOCKHEAD, 133, 141–142, 304
boundaries, 53, 222
Brooks, 25, 47, 199
bucket, holds the nodes for a particular
time-cell, 152
buffer slot indexes, 141

\( C^1 \) and \( C^2 \) functions, 79–81, 97
\( C^4 \), the set of first order changes
\( C^2 \), the set of second order changes
cell, 20, 52
square, 60
triangular, 60–62
cell parser, 60, 239
cellular topology, 20, 52, 228–232, 245,
250
cellwise-time, 244–254
CENTRAL-SYSTEM, 47, 139, 143–202
centroid, an object’s position, 78
Chapman, 37, 46, 70, 112, 115, 123, 133,
137, 143, 175, 211

childLink, 65
CITYTOUR, 27
CLOCK, 17
closed loop system, 3
COLLECT, 149
combining joint probabilities (CJP), 162,
168, 180, 269–270
common primitives, 74–78
compact-encoding, 5–8, 72
complexity
of DDN, 155

323
of spatial representation, 63
composite union, 54, 236
⊕, composition function, 165
⊙, string concatenate, 286
conduits, 82, 103, 134, 257–259
connection rules, 152
connectivity, 55, 224
contextual indexing, 54–69, 85, 111, 183, 206
continuity, 56, 95, 224
control and planning, 4, 38–50, 136–202, 208
DDN graph (DDNG), 155
diegetic representation, 31–33, 115–120
domain-layer, 11
Dretske, 30, 176, 191
dynamic decision network (DDN), 150–157
effector commands, 124, 144, 291
ego-centric, 112
entities
PB, the-path-blocker, 182
CA, the-conflict-area, 182, 186
S, the-stationary-vehicle, 182
the-other, 115
the-rejobj, 115
entity, 115
episode, 26, 71, 89, 244
event, 25, 71, 80
event reasoning, 71
event composition, 89–102, 188
event detection, 79–88
routines, 129
events and behaviour, 4, 25–37, 70–135, 207
examples
close the door, 79, 99
ter the roundabout, 91, 103
following, 130, 167
give way, 94, 132, 180
in the same region, 107
overtaking, 130, 167, 191
queueing, 95, 112, 132, 197
region crossing, 91
exocentric, 112
F, the set of frames, 79
f(t), a frame at moment t, 79
fblr α, a 4-tuple of type α, 78
fblr denotes front, back, left and right
field values, 32
field-of-view, 2, 13, 66
FINST, 43, 122, 147, 190
Fleck, 18–21, 52, 65, 211, 230, 232, 241, 244, 245
Fodor, 47, 137
frame of reference, 32
frame rate, 125, 201, 250
Garfinkel, 31, 39, 40
gate language, 276
GEM, 45
geographic information systems, 14
gestalt, 120–123
getHistory, 97
Gibson, 29, 36, 39
give way regions, 67, 68, 183
global-form, 112, 139, 147, 175, 189, 198
grouping, 120, 122, 190–191, 197, 202
Θ, heading difference function, 119
heading, 78
Heidegger, 34, 39
Heider and Simmel, 27–29
Heritage, 31, 39–40, 136
HIVIS, 1
HIVIS-MONITOR, 79–109
HIVIS-WATCHER, 115–202
functional description, 151
top level loop, 138
two stage DDN and TASKNET
system cycle, 160
hyperplanes, 23, 117, 233, 258, 259, 263
ignore, 154
incidence structure, 65, 231
inference-layer, 11
\( \mathbb{Z} \), the set of integers, 60, 78, 175
interval description tuple (IDT), 97
interval tree, 255–256
KADS, 11
Kalman, 36, 76, 259
KERNEL, 175–177
  arbiters, 184–185, 192–194
  operators, 127–128, 184, 192
lattice of tasks, 38–39, 149
leaf merge, 54, 236
leaf split, 54, 236
leafRegionParentTypes, 65
learning, 15, 18, 36, 40, 209, 210, 261
Lisp, 58, 160, 274, 276
local-form, 112, 115, 120, 139, 147, 175, 189, 198
lookup, 97
MACDDN, 153–155
MACNET, 9, 46, 143–146, 274–304
map file, 58, 238
marker, 36, 176
Markov, 49, 100
MERCATOR, 18, 210
Michotte, 27–29
model matching, 5, 19
NAOS, 27
\( \mathbb{N} \), the set of natural numbers, 233, 283
natural language, 25–27, 31, 70, 211
Neumann, 15, 27, 70, 100
NEWTON, 17
Newtson, 29, 71, 79, 129, 250
node tuple, 152
\(( \neg \alpha )\), \( \alpha \) is not present, 251
nowc, now-connections, 152
NS-symbol, Newell and Simon’s definition of symbol, 32
num**, the set of qualitative orientation values, 78
object, 4
object description tuple (ODT), 97
occupiedRegions, 78
official-observer, 4
open loop system, 2
operators, 46–48
arguments, 124, 291
  general definition (op4), 88
  OP, the set of operators, 88
  OP-execution, 123
optic array, 29, 111, 141
orientation, 78
orthogonal separation to input/central split, 47, 199
ostc, one-step-temporal-connections, 152
parentLink, 65
path planning, 15, 259
PENGI, 133, 143
PERIPHERAL-SYSTEM, 47, 123–128, 139
  operator processor, 123
PHOENIX, 45
plan recognition, 44–45, 100
point of action definition, 29
policy, 148, 174
polyhedron, 22
pose2ODT, 97
poseBox of an object, 78
poseBox2Cells, 78
positional feature, 117–119, 263
problems
  amount of evidence, 38, 43, 48, 147, 149
  blinkered (design feature), 137, 149
  boundary ownership, 53, 231, 232
  computational load, 147, 149
  different explanations, 38, 43
  episode completeness, 89
  event composition, 89, 97, 101
  frame problem, 96, 97
functional isolation, 137
loci of proximity, 235
spatial arrangements, 85, 112, 120, 129, 130
surveillance problem, 2–4, 13, 25, 38
viewpoint integration, 137, 147, 149
proximity, 121, 265
PRS, 45–46
putHistory, 97
quadtrees, 22, 62–64
$\mathcal{R}$, the set of real numbers, 35, 225, 226, 228
reflexive accountability, 39, 143
regions, 53
$R = LR \cup CR$, 65
CR, the set of composite regions, 65
LR, the set of leaf regions, 65
regions-a-cell-is-in, 65
regionType, 65
registrar, 144, 145, 291
regular cell complex or CW-complex, 230
retract, 16, 235
REX, 46
routines, 33–34, 40, 42, 46–48, 125, 129, 133
agency operators as micro routines, 125
Identify-head-and-tail, 192
Look-for-path-blocker, 183
Notice-starts-to-move, 183
Notice-stopping-object, 182
Notice-the-overtaking-pair, 192
Watch-for-enter-conflict-area, 183
Watch-for-switch-in-order, 192
Work-out-conflict-area, 183
running arguments, 46, 290
sbounds, surrounding bounding box, 64
scenario
enter, 103, 194, 213
occlusion, 167, 180, 213
queueing, 197, 213
SEQ, 89, 129
Serat-Frenet, 257, 259
SIM, 89, 129
simple and complex operators, 147
singly connected network (SCN), 162, 267
SONJA, 70, 209
spatial decomposition, 18, 54–55, 66
spatial representation, 4, 13–24, 52–69, 206–207
spatial-layout, 52–69
speed, 78
state, 176, 244
state-change, 176, 244
steady activity, 80, 244
strategic-layer, 11
subcomplex, 53, 233
Tarski, 23, 32
task based control, 147–149
task-layer, 11
TASKNET, 157–159, 165
TEA, 49, 201
temporal history builder (THB), 96–98, 255
$, a$-time-cell, 250
$\mathcal{I}$, a time-line boundary, 250
TMM, 44, 96
tolerance space, 74, 241
tracking-state, the state of an object at
time $t$, 78
traffic flow model, 142, 261
typical-object-model (TOM), 143
VIEWS, 5–8, 44, 58, 71, 102, 209, 211
virtual world, 141–142
watch, 154
weights, 101, 188, 272–273
$\bullet$, a DDN node “attribute wild card”, 152
wires, 124, 276–277
Yarbus, 41, 179, 200