Agents, Objects, and Events

A computational approach to knowledge, observation, and communication

René Ahn
CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN
Ahn, René Marie Charles
Agents, objects and events : a computational approach to knowledge, observation and communication / by René Marie Charles Ahn. -
Eindhoven : Technische Universiteit Eindhoven, 2000. -
Proefschrift. -
ISBN 90-386-0853-5
NUGI 855
Keywords: Artificial intelligence / Knowledge representation /
Semantics / Temporal reasoning / Type theory

IPA dissertation series 2001-02
© Copyright by René Ahn

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics).
Agents, Objects and Events
A computational approach to knowledge, observation 
and communication

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Eindhoven,
op gezag van de Rector Magnificus, prof.dr. M. Rem,
voor een commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen op
maandag 22 januari 2001 om 16.00 uur

door

René Marie Charles Ahn

geboren te Gronau
Dit proefschrift is goedgekeurd door de promotoren:
prof.dr. H. C. Bunt
prof.dr. J. C. M. Baeten

Copromotor:
dr. V. A. J. Borghuis
Acknowledgements

Writing this thesis would have been impossible had I not received the help and advice from many people.

I would like to thank my promotor, Harry Bunt, for his painstaking and patient reading of my writings, and for his many suggestions for improvements of the text. I also thank Tijn Borghuis, my copromotor, who always urged me to clarify my thoughts, and provided continuing enthusiasm and curiosity, as well as large amounts of valuable feedback and strong coffee. I am also grateful for the valuable comments and suggestions made by Jos Baeten, Don Bouwhuis, and Zhaohui Luo, the members of my thesis committee.

Over the years, many of my friends, colleagues and teachers have had decisive influences on my thinking, or have somehow helped me over one of the many hurdles, even though I did not always realise this at the time. I would like to mention Robbert-Jan Beun, Leen Helmink, Jan Jaspars, Leen Kievit, Hans-Peter Kolb, Emiel Krahmer, Kees Kuijpers, Judith Maethoff, Rob Nederpelt, Floris van Nes, Kees van Overveld, Paul Piwek, Rindert Schutten, Boudewijn Verhaar and Margriet Verlinden.

Finally, I want to thank all my other colleagues at IPO. Without IPO, this thesis would not have been written.
## Contents

1 Introduction
   
   1.1 Aim of this thesis ............................................. 1
   1.2 Motivation ....................................................... 2
   1.3 Information and computation ................................. 3
   1.4 Introducing the main problems ............................... 4
      1.4.1 Knowledge and reality ................................... 4
      1.4.2 Reasoning about time .................................... 6
      1.4.3 Communication ........................................... 8
      1.4.4 Summing up ................................................ 10
   1.5 Knowledge and reality ......................................... 11
      1.5.1 Possible worlds models .................................. 11
      1.5.2 An alternative approach .................................. 13
      1.5.3 Type theory as a conceptual calculus .................. 16
      1.5.4 Intensionality of concepts ............................... 19
   1.6 Reasoning about time ........................................... 20
      1.6.1 Model-oriented methods .................................. 20
      1.6.2 Logical methods ........................................... 21
      1.6.3 The frame problem ........................................ 22
      1.6.4 Our approach .............................................. 22
   1.7 Communication .................................................. 24
      1.7.1 The DenK project .......................................... 24
      1.7.2 Design of the interface ................................... 25
      1.7.3 Type theory as a semantic framework ................. 27
   1.8 Overview of the thesis ........................................ 28
      1.8.1 Summary of chapters 2-7 ................................. 28

2 Type theory .......................................................... 31
   2.1 What is a type system ......................................... 31
      2.1.1 Contexts ................................................... 33
      2.1.2 Typing rules ............................................... 33
      2.1.3 Terms ....................................................... 35
      2.1.4 Levels of the hierarchy .................................. 36
      2.1.5 Term construction ........................................ 37
CONTENTS

2.1.6 Some simple examples ........................................ 37
2.1.7 Propositions as types ....................................... 38
2.1.8 Assumptions in a context .................................. 39
2.2 Example .......................................................... 40
  2.2.1 Constructing a legal context .............................. 42
  2.2.2 Dependent function types ................................. 43
  2.2.3 Encoding quantification ................................. 45
  2.2.4 Checking proofs ........................................ 46
2.3 Pure Type Systems ................................................ 46
  2.3.1 Syntactic structures ....................................... 47
  2.3.2 The rules ................................................. 48
  2.3.3 Some properties ......................................... 50
2.4 Books and definitions ........................................... 51
  2.4.1 The use of definitions ................................... 52
  2.4.2 Some useful definitions .................................. 53
  2.4.3 Writing books ............................................ 55
2.5 Σ-types and inheritance ......................................... 56
  2.5.1 Σ-types ................................................... 56
  2.5.2 Inheritance .............................................. 57
2.6 The meaning of types ............................................ 59
  2.6.1 Specifications ............................................. 60
  2.6.2 Types as specifications ................................... 62
2.7 Specifications and realisations ................................ 63
  2.7.1 Extending segments ....................................... 64
  2.7.2 Top-down construction ................................... 65

3 Knowledge and observation ....................................... 67
  3.1 The problem .................................................. 67
  3.2 A book of knowledge ......................................... 68
  3.3 Example ....................................................... 70
    3.3.1 The logical layer ....................................... 71
    3.3.2 The mathematical layer ................................ 71
    3.3.3 The conceptual layer ................................... 72
    3.3.4 The factual layer ....................................... 74
    3.3.5 Drawing conclusions .................................... 75
    3.3.6 Summary ................................................ 77
  3.4 Semantic grounding ............................................ 77
    3.4.1 The problem ............................................. 77
    3.4.2 Gathering information ................................... 79
  3.5 Observation .................................................. 81
    3.5.1 Some definitions ....................................... 81
    3.5.2 Observable types ....................................... 82
    3.5.3 Example ................................................ 84
CONTENTS

3.5.4 Recognising individuals ........................................... 85
3.6 Intensionality ........................................................... 87
3.6.1 Nonexistent objects ................................................. 88
3.6.2 Coreferent expressions ............................................. 89
3.6.3 Ambiguous stimuli ................................................... 90
3.7 Knowledge state dynamics ............................................ 92
3.7.1 Resource-bounded reasoning ....................................... 93
3.7.2 Observation ............................................................ 94
3.7.3 Revision of beliefs ................................................... 95
3.8 Conclusion ............................................................... 96
3.8.1 Going on ............................................................... 97

4 Modelling dynamic objects ............................................. 99
4.1 The problem of change .................................................. 100
4.1.1 Conventional approaches .......................................... 100
4.1.2 The root of the problem ............................................. 101
4.2 A structural view of change .......................................... 103
4.2.1 Examples of localisation ............................................ 106
4.3 Ontology ................................................................. 108
4.3.1 A catalogue of categories .......................................... 109
4.3.2 Looking back ......................................................... 114
4.4 The logical framework ................................................ 115
4.4.1 Representation language: syntax ................................ 115
4.4.2 Representation language: semantics ............................ 117
4.5 How to reason ........................................................... 118
4.5.1 Knowledge about structure ....................................... 119
4.5.2 Knowledge about events .......................................... 121
4.5.3 Knowledge about time ............................................. 122
4.5.4 Knowledge about particulars ..................................... 124
4.6 Reasoning example ..................................................... 125
4.6.1 Crossing the border ................................................ 125
4.6.2 Painting .............................................................. 128
4.6.3 Painting a room ..................................................... 130
4.6.4 In the mood .......................................................... 133
4.7 Conclusions and discussion .......................................... 134

5 Time, types and memory ................................................ 139
5.1 Adaptation of the knowledge state .................................. 139
5.1.1 Reinterpreting the conceptual layer ............................. 140
5.2 Extensions of the knowledge state .................................. 143
5.2.1 Time ................................................................. 144
5.2.2 Time slices .......................................................... 145
5.2.3 Expressing facts ...................................................... 147
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.4</td>
<td>More about time</td>
<td>148</td>
</tr>
<tr>
<td>5.3</td>
<td>Observation and reasoning</td>
<td>150</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Observation example</td>
<td>151</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Reasoning example</td>
<td>152</td>
</tr>
<tr>
<td>5.4</td>
<td>Past, present and future</td>
<td>154</td>
</tr>
<tr>
<td>5.4.1</td>
<td>About the future</td>
<td>155</td>
</tr>
<tr>
<td>5.5</td>
<td>Conclusions</td>
<td>155</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Going on</td>
<td>156</td>
</tr>
<tr>
<td>6</td>
<td>Type theory and DRT</td>
<td>157</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>157</td>
</tr>
<tr>
<td>6.2</td>
<td>Discourse Representation Structures</td>
<td>159</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Syntax</td>
<td>159</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Truth conditions</td>
<td>160</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Examples</td>
<td>162</td>
</tr>
<tr>
<td>6.3</td>
<td>Translation of examples</td>
<td>164</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Example 1</td>
<td>164</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Example 2</td>
<td>166</td>
</tr>
<tr>
<td>6.4</td>
<td>Formalising the translation</td>
<td>166</td>
</tr>
<tr>
<td>6.4.1</td>
<td>The translation function T</td>
<td>167</td>
</tr>
<tr>
<td>6.5</td>
<td>Semantics</td>
<td>170</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Simulating 'classical' semantics</td>
<td>170</td>
</tr>
<tr>
<td>6.5.2</td>
<td>The advantages of subjective judgement</td>
<td>173</td>
</tr>
<tr>
<td>6.6</td>
<td>Conclusions and Discussion</td>
<td>174</td>
</tr>
<tr>
<td>7</td>
<td>Inter-agent communication</td>
<td>175</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td>175</td>
</tr>
<tr>
<td>7.2</td>
<td>Setting the scene</td>
<td>177</td>
</tr>
<tr>
<td>7.2.1</td>
<td>The common vocabulary</td>
<td>177</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Coding and decoding</td>
<td>178</td>
</tr>
<tr>
<td>7.2.3</td>
<td>Common knowledge</td>
<td>179</td>
</tr>
<tr>
<td>7.2.4</td>
<td>Messages</td>
<td>182</td>
</tr>
<tr>
<td>7.2.5</td>
<td>Processing of segments</td>
<td>183</td>
</tr>
<tr>
<td>7.3</td>
<td>Communication</td>
<td>184</td>
</tr>
<tr>
<td>7.3.1</td>
<td>The situation</td>
<td>185</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Providing information</td>
<td>186</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Seeking information</td>
<td>189</td>
</tr>
<tr>
<td>7.4</td>
<td>Applying the mechanisms</td>
<td>190</td>
</tr>
<tr>
<td>7.5</td>
<td>Conclusions</td>
<td>192</td>
</tr>
</tbody>
</table>
CONTENTS

8 Conclusions 195
  8.1 Discussion ................................................. 195
  8.2 Future research ............................................. 198
    8.2.1 Reflexivity ........................................... 198
    8.2.2 Concept learning ...................................... 199

A A robot book 211
  A.1 Preliminaries .............................................. 212
  A.2 Axioms about time ........................................ 213
  A.3 Concepts and invariants in the robot world ............... 214
  A.4 Observational data ...................................... 215
  A.5 Conclusions .............................................. 216

Samenvatting 218

Curriculum Vitæ 220
Chapter 1

Introduction

1.1 Aim of this thesis

This thesis is concerned with the following question: how can we model the information of an intelligent agent about a changing external reality in a computational way?

Throughout, we take a computational viewpoint on information, and assume that the information of an agent can be represented in some dynamic computational structure, a *knowledge state*. We assume that the knowledge state of such an agent is:

- **private and subjective:** it is not visible from the outside, and it is given in terms of concepts that are not necessarily universal, but are at least in principle personal to the agent himself.

- **realistic and useful:** the agent’s information is semantically grounded in the outside world. The agent uses this information to draw conclusions that go beyond his immediate observations. More in particular, the external world is subject to changes, and the agent is able to reason about the effects of supposed or actual events in the external world, to make predictions about the future, or reconstruct past events.

- **dynamic:** the knowledge state of the agent can change through inference or belief-revision processes, that are internal to the agent, or through communication and observation processes, that involve the agent’s environment.

We are concerned with the representation of this information, its computational realisation, its dynamic nature, and the nature of the relationship between this information and the environment of the agent. More specifically, we want to construct a model of an agent’s knowledge state that shows how the agent can reason with his information, how this information can be
meaningfully related to an external reality, and how it can be extended as a result of observations. As the external reality is subject to changes, we also have to explain how the agent can organise his knowledge in such a way as to be able to combine information pertaining to different moments in time, and how he can reason about the effects of external events. Finally, we want the model to indicate how an agent will be able to communicate with other agents, despite the private nature of his knowledge state.

Psychological realism is not one of our goals. Instead, we try to satisfy the more modest goal of computational realism. We wish to make sure that we understand how the agent can function from a computational viewpoint, i.e. in terms of effective procedures\(^1\), taking into account the personal subjective viewpoint of the agent, and his limited access to information. For if we are to be serious about computational realism, we have to indicate how the agent, and not some omnipotent external observer, might be able to perform the computations needed to consult and update the agent’s information state.

### 1.2 Motivation

Why do we consider these questions? There are both practical and theoretical reasons:

1. The practical reason is that it would be useful to build agents that can act as intelligent assistants to humans that are working in different fields. This is especially important for users that interact with increasingly complicated computer applications. In these cases the ‘external’ reality is itself a part of cyberspace, i.e. it is an electronic and virtual reality, which users have difficulty in observing, understanding and manipulating. Such users could do with some assistance. An artificial agent that is ‘at home’ in the electronic environment could offer such assistance. Many of the ideas that are presented in this thesis have taken form while the author was working on the ‘DenK’-project, whose main goal was to uncover the knowledge needed to build such electronic assistants, [Ahn & al., 1994]; [Bunt, 1998].

Indeed, the construction of such agents involves the development of a comprehensive and integrated model of knowledge states, in which the relationship between an agent, his information, and the changing external environment can be studied in a computational way.

To be able to construct such an agent, it is not sufficient to model the knowledge state of the agent in isolation. We also need to understand how his knowledge state can be semantically grounded in an external

\(^{1}\) In the spirit of the Church-Turing thesis
1.3. INFORMATION AND COMPUTATION

reality (or cyberspace), and how the agent's knowledge might grow as a result of observations. Moreover, the model should explain how the agent can reason about the properties and dynamics of the objects that surround it. Finally, it should be clear how the agent can communicate, preferably in a language that resembles natural language, and which respects the subjective and private nature of the agent's information state. In a practical setting, it is clearly crucial that the model that we construct is computational. Only then can we hope that it is useful.

2. Of course, the problems that we consider also have quite some theoretical interest. In fact, we touch on matters that are central in philosophy, semantics, cognition and artificial intelligence. Agents have information about an external world. They are able to reason and to communicate, and they can interpret their observations in terms of this knowledge. Numerous well-known philosophical and semantical problems arise when one tries to formalise this situation. To name but a few: the distinguishability of coreferent expressions, the meaning of empty categories, the identity of objects across time, the logical treatment of change, the problem of logical omniscience, the nature of presuppositions, and the meaning of questions. The construction of a computational model in which reasoning, observation and communication are integrated can shed some new light on these matters. For such a model does, by its very nature, offer possible solutions for some of these problems, and also shows that these solutions are compatible. In fact, one might argue that a computational model that integrates an account of knowledge, observation and communication does amount to a (partial and primitive, yet computational) theory of meaning for the language that such agents use as a vehicle of communication.

1.3 Information and computation

If the computational realisation of information is taken seriously, various questions arise. For example, we may be concerned with efficiency or with the formal complexity of algorithms. But, in this thesis, we focus on the question whether certain information can be computed by an agent in principle. The point that we want to stress is simple: if an agent is supposed to use certain information, he has to be able, at least in principle, to obtain this information. Though there is a sense in which this constraint is extremely obvious and practical, it is actually the case that many theories, particularly current semantic and epistemic theories, do not take this constraint into account. They do not distinguish information that is available, or that can be computed by an agent, from 'facts' about which the agent has no, or insuffi-
cient, information. Theories that lack computational content simply cannot be implemented.

But though practical matters are important, the most important reasons to worry about computational aspects of information are, in fact, methodological and theoretical ones. For when we try to understand the cognitive abilities of agents, we are dealing with abilities that some agents, be they robots, cats, or humans, really exhibit. We may therefore assume that there exist causally effective mechanisms that ‘implement’ these cognitive abilities. These mechanisms allow agents to compute the knowledge that they are said to have. If we ask ourselves how the agents succeed in computing their knowledge, given their (necessarily) subjective view on things, we are forced to consider the whole situation from a rather different, and more pragmatic perspective. This perspective may well hold the key to the solution of certain problems that haunt present-day semantic theories.

1.4 Introducing the main problems

To introduce the approach that we take in this thesis, we will, in this chapter, consider the three main issues that will be addressed. These are:

- the modelling of knowledge, including its relation to the external world.
- the modelling of time and change.
- the modelling of communication processes.

This short list leaves out many other important questions, that we have neither time nor place to address here: we will say very little about action, and will not touch on (inductive) learning, or on the matter of reflexivity of thought and language. However, we will try to indicate, in the final chapter, how some of these questions might be approached in ways that are compatible with the model that we will construct.

1.4.1 Knowledge and reality

Most theories of knowledge are ultimately extensional in nature. The hallmark of such theories is that they use models that reflect the external world in terms of objects. The objects are assumed to be given, and can stand in various relationships, that are also given. Formally, the relationships between objects are modelled in terms of sets: sets of objects, sets of pairs of objects, etc. Equality of sets rests on equality of their elements, and eventually bottoms out in equality of the objects. Whether objects are equal is assumed to be given. This idea, which works quite well in artificial domains, like
mathematics, has its drawbacks if we want to transfer it to the real world, and apply it to the relationship between an agent and his surroundings. An agent that tries to understand the world in terms of objects, has to perform quite a computational feat before he can even start to do so. Also, the resulting interpretation is a subjective one, and not necessarily without errors. This implies, among other things, that the notion of equality between objects, which is assumed to be basic in extensional theories, is rather problematic for real world agents. Though people try to understand the world in terms of objects, they may well have considerable difficulty in recognising or distinguishing them.

So, looking at extensional theories from a practical and realistic viewpoint, we see that they relate a knowing agent to a world that is already interpreted and conceptually known. These theories apparently presuppose an all-seeing external observer that can inspect both the external world and the knowledge state of the agent, and is able to relate the two in some way. As far as we can see this holds for theories that involve possible worlds, e.g. [Kripke, 1972], [Montague, 1974], [Hintikka, 1962], as well as, for instance, situation theory, [Barwise & Perry, 1983]. These theories understand the external world in terms of the concepts of an observer that is placed outside the theory, and whose judgement one is not willing (or indeed able) to question. Such an approach which is dubbed 'extrinsic epistemics' in [Benthem, 1991] is therefore not entirely realistic and sidesteps important computational questions. To create a form of 'intrinsic epistemics', a more computational account of knowledge is called for, that takes the viewpoint of the knowing subject into account. Such an account should relate knowledge to such basic things as observation and recognition.

For this reason, we will, in this thesis, dispense with the external observer and his concepts altogether. We want to understand the knowledge representation problem from the subjective viewpoint. The resulting view should express that the knowledge of an agent about an external world is not simply true or false, but is always based upon a conceptualisation of the external world, a conceptualisation that is fallible, that is computed by the agent himself, and that is personal to the agent. If this idea is accepted, it is immediately apparent that the concepts that an agent uses to understand the world are central to his knowledge state. To understand how an agent 'knows' the world, we then have to develop a view that explicitly includes both the world as it is given, and the conceptualisation that an agent employs in order to grasp this world. The resulting view is really a two-sided one. On the one hand, there is an external reality, that consists of the things that are, but which are, in themselves, uninterpreted, and a priori meaningless. On the other hand there is an agent, who is in the trade of making judgements about the patterns that are discernible in the reality that he is confronted with, and carves up this reality in terms of his own concepts and
CHAPTER 1. INTRODUCTION

categories, projecting his own meanings into the reality surrounding him. To understand this picture, we will be forced to model a process like observation in such a way that it becomes clear how an agent can extend his ‘knowledge’ through observations, while, at the same time, this knowledge is given in his own subjective categories and concepts.

Note that these ideas lead to a view on knowledge\(^2\) that is rather different from the received one. For instance, knowledge can no longer be ‘true’ in any absolute sense. This does not make it useless or nonsensical. Knowledge can still be reliable and trustworthy, and it can be agreed upon by different agents that use similar conceptual systems. However, in this computational picture, all knowledge, even the most reliable kinds of knowledge, risks to be refuted by new observations.

Therefore, the difference between knowledge and belief now seems to be largely a matter of degree and turns into the social phenomenon that many of us ‘know’ it to be. On a more positive note, knowledge now becomes more open and extensible. An agent may always learn new concepts, which increase the scope of his conceptualisation of the world, thereby opening whole new vistas.

Formally, we will develop this view through the use of type theory\(^3\) (TT) [Barendregt, 1991],[Coquand, 1985],[Martin-Löf, 1984]. As we will see, type theory is a formalism in which concepts can figure as first-class citizens. This allows us to be precise and explicit about the relationship between concepts and their ‘instances’ or ‘justifications’, and, above all, reminds us that concepts have a life of their own, independent from their ‘instances’.

1.4.2 Reasoning about time

An important complicating factor in the modelling of knowledge is the fact that the external world is not static, it changes. Therefore, if an artificial agent is ever to be useful in any environment which is unlike pure mathematics, it is vital that this agent has the ability to reason about time, events, and the changes that events can bring about.

This is difficult. As we will see, the basic problem is to specify how the propositions that hold after an action are related to the propositions that hold before the action. In fact this problem, often called the ‘frame’ problem, has become quite notorious in Artificial Intelligence, and is widely discussed, also among psychologists and philosophers. See, for instance, [Haselager, 1995], [Dennett, 1987]. Fortunately, we do not have to solve this problem in general.

\(^2\)Here we take knowledge in the everyday sense of the word. We quote the definition in Webster’s new dictionary of synonyms (p. 481): ‘Knowledge applies not only to a body of facts gathered by study, investigation, or experience but also to a body of ideas acquired by inference from such facts or accepted on good grounds as truths.’

\(^3\)Chapter 2 contains a short introduction to type theory.
1.4. INTRODUCING THE MAIN PROBLEMS

We will, instead, consider the far simpler problem of reasoning about discrete event systems. These systems are still quite general, and are well suited to model changes in common user-appliances or in cyberspace.

Also, we have a trick upon our sleeve. To formally relate the propositions that hold after an action to the propositions that hold before the action, we devise a new method. This method rests on a simple, but important observation: that events only affect propositions in an indirect way. This is best explained through an example. Suppose John is working with his computer. Consider the following proposition: “The window of the internet browser is the biggest window on John’s screen”. Call this proposition: ‘P’. John clicks on his mouse to minimise the browser window, and ‘P’ no longer holds. Call the mouse-event: ‘e’. Clearly, the mouse-event ‘e’ has affected the truth value of the proposition ‘P’. But why has this event affected this proposition? It is obvious that the event has changed the layout of John’s computer screen. More precisely, the event has somehow changed the state of some particular object, the state of John’s browser window. Actually, it has changed only one aspect of the state of this window, its size.

This simple observation brings out that the event ‘e’ has not affected the proposition ‘P’ in any direct way. The event has only affected this proposition because it has changed a particular aspect of the state of a certain object. Generalising, we see that an event will only affect those propositions that depend on the state of the structures that this event has changed. This is important because the structure-changing effects of an event are local. Events only change a limited number of objects, and they also only change certain aspects of these objects. Those propositions that are changed by an event must all depend somehow on the state of a particular local structure that this event has changed. For instance, take the proposition ‘Q’: “The window of the editor is the biggest window on John’s computer screen”. If it turns out that ‘Q’ is true after the mouse-event ‘e’ has happened, then we can understand how this came about. Though the event ‘e’ has not done anything to the state of the edit window, as it has only affected the browser window, we understand that the truth of the proposition ‘Q’ depends on the sizes of the windows on the screen. And one of these sizes has been changed. In fact the event ‘e’ has changed the truth value of countless propositions, but it will have done so through the rather limited and localised state-change that constitutes the essence of this event; a change in the size of the browser window. This suggests that it may be possible to reason about the effect of events in cyberspace on the basis of information of the state changes that they bring about. This is what we will do.

There are at least two reasons to assume that such an approach may be advantageous:

- As we have seen, the underlying state changes can be localised, and are
quite limited. We can, at least in principle, indicate what changes and what does not change due to an event.

- Properties and propositions are higher-order concepts, while the things that are being changed (like sizes of windows) are first-order concepts. If all of our changes can somehow be confined to first-order concepts, this will decrease the complexity of the theories that describe the changes. For there are far more propositions over a given set of objects than there are objects.

In order to develop this line of thought, we first have to gain a proper understanding of the way in which propositions can depend on the states of objects. To do so, we must allow objects to change. This means that we have to clear up what seems to be a significant conceptual confusion. Conventionally, objects are often treated as if they are are unchangeable mathematical values, like, for instance, numbers. Accordingly, their identity is treated as if it were just a common form of identity. But, as we will see in chapter 4, changing objects are not ‘simple’ mathematical values, and the notion of identity for such objects has to be carefully analysed in order to be properly understood. Actually, we will identify three different equivalence relations that play a role in the understanding of objects and their changes. If these identities are distinguished, the relationship between objects and their states can be clarified, and the logical paradoxes that tend to result from the combination of information about different moments will disappear. In this way it is possible to give a — rather satisfying — treatment of time and change, while staying within the confines of standard logic.

1.4.3 Communication

A third phenomenon that concerns us is communication. An agent that can observe an external world does not necessarily observe the same things that other agents observe. If he has the ability to communicate, he can exchange information with others. If this process is successful, his knowledge will grow. As a communicating agent can also pass on information to third parties, a form of ‘collective information dynamics’ will emerge, enabling agents to deal with information from afar, thereby greatly empowering the agents.

To understand the communication process, we have to know how messages that are sent and received may affect the knowledge states of agents in such a way that information can be exchanged. Clearly, this problem is quite complicated in the setting where the information in the knowledge states of

\footnote{Unfortunately, it is difficult to assess whether information that one gets from others is reliable. As a result, the pool of collective information may be contaminated with disinformation, which reduces its usefulness.}
different agents is given in terms of concepts that are, at least in principle, private to the different agents. In such a setting one can no longer assume that agents that communicate somehow share large name spaces that enable them to communicate the identity of objects in a direct way.

To tackle the problem of how information does get exchanged, one has to choose a language that the agents use to communicate. It is quite difficult, and perhaps impossible, to make this choice in a principled manner. In this thesis, we will investigate this question against the traditional background of natural language communication. There are three reasons for doing this:

- Since people use natural language, it would be very helpful if we can equip an artificial assistant with communication abilities that reflect the expressive spectrum that natural languages tend to offer.

- The model that we present in this thesis has largely been developed in the context of the DenK-project. [Ahn & al., 1994] This project aimed at an understanding of knowledge and communication in computational terms, in order to build man-machine interfaces that use natural language.

- As it turns out, the model of private knowledge states that is proposed here, sheds a new and interesting light on certain phenomena that are familiar from the study of natural language communication, like the modelling of questions, and the use of determiners.

Of course, natural language semantics and pragmatics are disciplines in their own right. To be able to discuss the exchange of information between type-theoretical knowledge states against the background of present day linguistics, we will adopt the following strategy. We will first show how one can construct a bridge between natural language utterances and formal operations on type-theoretical contexts. To do so, we take an existing linguistic theory (DRT) and relate this to an approach where utterances are understood as 'context changing operators', applied to the type-theoretical knowledge states of the agents. We then go on to investigate a natural situation in which two agents that are endowed with type-theoretical knowledge states are trying to communicate. This is difficult, as their knowledge states are private, and information can only be exchanged if it is somehow expressed in words.

The problems that arise in this situation are clearly relevant to a proper understanding of communication processes. People also have private knowledge states, and the name space that they share, enabling them to refer to individual objects, is very restricted. Most objects in the world, like chairs, pencils and shoes, do not have any names. Accordingly, agents have to use rather complicated strategies to try and refer to them in understandable
ways. This has also been observed in experiments: [Grosz & Sidner, 1986], [Cremers, 1996]. Though our model of communication is highly simplified, it does show how information can be exchanged successfully, and it does help us to understand the function of determiners, and the interplay between questions and answers. The model shows that such well-known linguistic phenomena may contribute to successful exchange of information as a result of the same underlying computational type-theoretical mechanism.

Our aim is to illustrate how a formalism like type theory, that focusses not on truth, but on the computational processes through which information is obtained, can be put to use to model communication processes. As is also clear from the work of [Ranta, 1994], [Ranta, 1991], [Piwek, 1997], and [Krahmer & Piwek, 1999], such an approach may be fruitful. Eventually, we hope to stimulate the development of a theory of communication that can be integrated with theories of reasoning and observation.

1.4.4 Summing up

In short, the approach taken in this thesis can be summarised in the following three points:

- We take a direct and computational approach to the modelling of knowledge, using a framework in which concepts figure as first-class citizens, and are emphasised at the expense of ‘individual objects’ or ‘entities’. We subsequently relate conceptual knowledge to an external world through an explicit account of observation and recognition.

- We take a physically oriented approach to the understanding of time and change, accounting for changing properties through the structural effects of events on the states of objects. To do so, we develop an ontology that reflects the relationship between events and state changes of objects.

- We model communication in terms of the dynamics of the knowledge states of speaker and listener. Given the private nature of these knowledge states, important (computational) problems in communication that are related to the private nature of knowledge are brought in the open, and can be investigated.

In the rest of this chapter, we proceed to discuss the three points given above in some more detail, and investigate how they relate to the different solutions that are found in the literature:

- We will first look into the relationship between knowledge and an external reality. In particular, we will consider the ‘possible worlds’ ap-
proach and explain why a more direct and computational approach, using a formalism like type theory might be preferable.

- Next we will consider the problem of reasoning about change, we give a (short) overview of the existing approaches to the problem, and indicate what we consider to be the main advantage of our approach.

- Finally, we introduce the approach to communication that is taken in the DenK project, and consider questions that arise when knowledge states formalised in type theory are taken as a semantic background of natural language interpretation.

1.5 Knowledge and reality

Knowledge is always knowledge about something. Theories of knowledge are haunted by the problem of how to model that elusive aspect of `aboutness', called `sense', which Frege [Frege, 1892] calls `der Sinn', an aspect which cannot be rendered in simple referential logical models of language. Frege's own example is still illustrative. Consider the expression: `The morning star'. As the reader knows only too well, this expression does not refer to a star but to the planet Venus. The same is true for the expressions `Venus' and `The evening star'. So all these expressions refer to the same thing. However, it is not the case that these expressions all have the same meaning. For a sentence like `The morning star is Venus' means something different from the sentence `Venus is Venus'. A person might deny the first sentence, but no one would deny the second one. Apparently, there is some difference in meaning between the expressions `Venus' and `The morning star'. The problem is how one can account for this difference.

1.5.1 Possible worlds models

A popular solution to this problem is through models that include so-called possible worlds [Kripke, 1972] [Hintikka, 1962]. Possible world models make it possible to distinguish between Venus and the morning star while, on the other hand, allowing it to be true that Venus is the morning star. The trick is to represent the knowledge of a certain agent by a collection of possible worlds, with an accessibility relation between them. The intuition behind possible worlds is that they represent the agent's epistemic alternatives, i.e. that they represent the different ways in which the world might be, for all that the agent knows. In some of these worlds and in particular in the actual one, the morning star is Venus, in others it is not. In this way, the difference in meaning between both sentences can be captured: two expressions only
mean the same thing in the actual world, if they refer to the same objects in all possible worlds that are accessible from the actual one.

As such models are enormous in size, it is impossible to use them in practice. This problem is somewhat alleviated by the fact that one does not have to use the model itself, but can use some form of inference instead. This means that one has to do inference in a modal logic, i.e. a logic with an extra (modal) operator. See, for instance, [Moore, 1995].

It can be shown [Hughes & Cresswell, 1984] that, under certain conditions, the set of possible worlds with their accessibility relation acts as a model of certain modal logics. More precisely, different axiom schemes for the modal operator correspond with different properties of the accessibility relation. These observations have led to various axiom schemes for modal logics that can be used to reason about knowledge and belief\(^5\).

This leads to formal systems of considerable power. For instance, as shown in [Halpern & Moses, 1990], using different modal operators to distinguish between different agents, it is possible to give mathematical characterisations of distributed and common knowledge that can be employed to analyse the effects of communicative actions. In this way one can design programs that implement protocols by regarding processors as agents. Also, as in [Moore, 1980], one can try to combine various modalities in order to describe both knowledge and action.

In the way we have indicated above, possible worlds theories are supposed to be able to model whether an agent that is placed in some actual world 'knows' or 'believes' certain things. For instance, the agent believes something to be true if it is the case in all worlds that are accessible from the actual one. The beliefs of the agent are true or false. They are true if they accord with the way things are in the actual world.

Such a viewpoint is attractive for many reasons. Among other things, it offers a firm footing. There is a way in which things really are. This means, for instance, that it is always clear (at least in principle) whether an agent is right or wrong. Also, as these theories are referential, this view is appealing due to its 'concreteness'. One can, for instance, model the meaning of a concept like 'red' in terms of the set of all red objects, a set that is somehow already given and really somewhere 'out there' in the universe of possibilities. But though the idea of possible worlds is appealing, it also has its difficulties.

First, it cannot deal very well with the meaning of expressions that do not just have no reference in the actual world, but which cannot have a reference. This is the case for expressions that denote logical impossibilities, like: 'The greatest prime number'. It is hard to imagine how one can represent, in

\(^5\)These ideas are not limited to epistemic modalities, but can also be applied to other modalities, like necessity, obligation, or even to temporal ones like 'it is always the case that', or 'it will eventually be the case that'.
1.5. KNOWLEDGE AND REALITY

terms of possible worlds, the knowledge that an agent gains when he learns: ‘There is no greatest prime number’.

A second problem, which may be somewhat more difficult to appreciate, is connected to the perspective of epistemic theories that are based on the possible world paradigm. For what do these theories actually model? It seems fair to say that what they model is an agent’s knowledge. But the model does not just include those things that the agent knows, but also has to include all those things that the agent does not know. For instance, the model includes, among others, the entire actual world. This is a consequence of the fact that these models look at the knowledge of agents from an external perspective. They model the knowledge of a given person as it appears to some objective, all-knowing outside observer. This is problematic because it tells us little about the knowledge state of the agent in question from the inside. In particular, it does not permit us to build an emulation of such an agent. It does allow us, in principle, to build an emulation of the all-knowing observer. But such attempts are bound to fail, because this illustrious observer knows far more then any of us will ever know.

The third problem of possible worlds theories is that they can only link the knowledge of an agent to the actual world if this actual world is assumed to be given together with a preferred interpretation. The world is supposed to be constituted in a certain way, and it is also supposed to be only understandable in a certain way. This consequence of the possible world theory is actually the most perplexing, and also the most unacceptable one. For even if the world really were constituted in a certain way\(^6\), its constitution is most certainly unknowable to us. Besides, even if there is a proper way to understand the world, it is rather obvious that people, throughout history, have understood the world (albeit imperfectly) in many different, and even in incompatible ways. And surely a theory of knowledge and meaning should be applicable to all, and not only to those whose knowledge is beyond reproach.

The fourth and final problem of possible worlds theories is that they simply fail to answer the most important question of all: how the knowledge of an agent relates to reality itself. But this is a question that can never be answered using ‘isolated’ mathematical models. To answer this question one has to relate an agent to an external reality.

1.5.2 An alternative approach

To cope with these difficulties the move that we propose in this thesis is a radical one. We will abandon referential theories of meaning that rely on an all-seeing observer altogether. Instead, we want to construct a theory that is

\(^6\)This would have rather interesting consequences. For instance it would mean that such questions as: ‘Are there really any electrons?’ would have a definite answer.
truly computational.

Considering the relationship between knowledge and reality from a computational viewpoint, we have the following problem: on the one hand there is an agent, that has somehow been able to condense his experiences into a body of knowledge, and on the other hand there is an external world that this knowledge is about. The problem is to see how the two may be linked in a computational way.

Being agents ourselves, we can get a feeling for what is involved here, by consulting our own experience. Suppose you arrive in a certain town for the first time in your life. Your eyes go over the scenery. Though there is nothing there that you have ever seen before, you can still recognise the various kinds of things. You see that there are different people, buildings, cars and traffic lights. Not because you have seen any of these particular individuals before, but because you recognise these things as instances of abstract concepts, concepts that reflect your previous experience in the world.

Once you have interpreted the given scene using some of these concepts, you can start to apply the general knowledge that you have about cars, people and traffic lights to the given situation. This may subsequently allow you to draw certain conclusions about this situation. Cars will remain on the streets, and stop at red lights, pedestrians will remain on the sidewalks, things will not move through walls, etc.

It seems, then, that you are able to connect your knowledge to an existing reality, because this knowledge is centered around concepts whose instances you can somehow recognise in the external world. You interpret your observational data in terms of these concepts. You ‘fit’ the world that you encounter into your conceptual scheme of things, in order to make observations.

We want to take this picture seriously, and hope to gain a computational understanding of the relationship between knowledge and reality for a given agent, by modelling this process in some detail. Let’s first look at recognition.

**Recognition**

What does it mean to recognise something in the external world, computationally speaking? To recognise something means that some computational criterion is satisfied by the external world. If the recognition criterion is fulfilled, an instance of the concept that goes with this criterion has been recognised. For a given agent, some of his concepts are observable, i.e. they are linked to observational criteria. As the agent can really apply these criteria, they must be computational. From the fact that an agent can apply a computational criterion to recognise a particular concept, one should not infer that the agent himself is necessarily able to analyse this criterion. The agent will be able to decompose criteria that correspond to complex concepts,
1.5. KNOWLEDGE AND REALITY

but this decomposition ends in concepts that are atomic\(^7\) to the agent.

What does the agent learn when a computational criterion that is associated with some observable concept is satisfied in the external world? The agent learns that there is 'something' in the external world that satisfies the given criterion, i.e. that there is an instance of the concept. At the same time, this is all that the agent learns. Relative to the agent, the 'things' in the external world that he can observe are highly 'encapsulated' from an information point of view. Most of the information about these external things is not available to the observer. The agent only knows that there is this 'thing' that satisfies the criterion: he does not necessarily know, for instance, what the identity of this thing is. If, making some new observation, the criterion is met again, the agent is normally not in a position of saying whether this is due to the same 'thing' that led to his first observation. The identity of 'things' is not a basic notion from an observational viewpoint. The basic notions are the 'concepts' that the observations produce an instance of. So, if we model observation in terms of recognition, in the way that we indicated, we see that the agent is confronted with what is essentially very fragmented information about the external world. The agent only gets anonymous pieces of a giant jigsaw puzzle, and he has to somehow put them together.

Background knowledge

To be able to put the pieces together, the agent has to integrate them within his current world view, he has to fit in these observations in his given background knowledge. This is not an entirely hopeless task, as the observations are based on criteria that reflect the concepts that the agent uses himself, i.e. the same concepts as those around which the background knowledge of the agent is constructed. To model this process mathematically, we need a formalism that allows us to write expressions that stand for the different concepts and their instances, that allows us to calculate with them, and that regulates the way in which observations of these concepts can be integrated in an overall structure that captures the agent's background knowledge.

The formalism that we choose for this task is type theory. As we will see, type theory is a calculus that allows us to represent both the concepts of an agent as well as his background knowledge, and enables us to relate the two. The expressive power of the formalism is considerable. When this formalism is extended with a simple account of the observational abilities of an agent, it provides us with precise rules to answer the following questions:

\(^7\)The criteria that correspond to atomic concepts are not analysable on the level on which the agent himself is able to reason. If an outsider analyses the agent and his cognitive activity, this outsider may be able to analyse some 'atomic' criterion that the agent uses, and understand that the agent is actually using a neural network or a parsing algorithm to recognise the instances of this atomic concept.
1. How to represent the background knowledge of an agent.

2. Which complex concepts can be formed, given the atomic concepts in the background knowledge of the agent.

3. How the agent can decide whether concepts are equal.

4. How the result of observations can be employed to extend the background knowledge of the agent.

5. How to draw conclusions by combining observed information with background knowledge.

6. Which concepts that can be formed, given the agent’s background knowledge, are observable in the external world.

An additional advantage of type theory is that all information is accompanied by justifications that allow us to keep track of the way in which information has been computed. Among others this means that we can always inspect the background knowledge of the agent to determine which information was involved in drawing certain conclusions.

In the next section we will indicate how one can use type theory to model the background knowledge of an agent, as well as the concepts and instances of concepts that play a part in this background knowledge. It will become clear how the above questions can be answered.

1.5.3 Type theory as a conceptual calculus

Type theory is a formal system. As such it consists of formal expressions that can be divided into a few syntactic categories. Tokens of these syntactic categories can stand in specific formal relations.

Categories of type theory

The principal syntactic categories that figure in type theory are:

- *types*: These we will use as representations of the ‘concepts’ of the agent. The concepts that can be formed are rather complex. Apart from concepts that classify simple things, like: ‘car’ or ‘person’ it will be possible to form concepts that classify whole states of affairs, like: ‘A person in a car’ or ‘A city in which every person has a car’.

- *objects*: These we will use as representations of the ‘instances’ of the various concepts. Instances can be atomic (those that stem from observations) or they can be composed from other instances (those that stem from deductions).
1.5. KNOWLEDGE AND REALITY

- **kinds**: At present, we will only deal with one single kind, written as: ‘*’. Its function will be explained below.

- **contexts**: These we will use as representations of the ‘background knowledge’ of an agent. As the agent will normally obtain new knowledge through observations, these contexts tend to grow.

**Inhabitants**

The principal relation in type theory is the *inhabitant* relation. The inhabitant relation is expressed in a *statement*, written as:

\[ X : T \]

This expresses that ‘X’ is an inhabitant of ‘T’. Objects are inhabitants of some type, and types are inhabitants of the kind ‘*’.

**Contexts**

Contexts are sequences of *declarations*. A declaration is a statement \( x : T \), where \( x \) is a variable. There are variables that stand for objects, and there are variables that stand for types. A context is called ‘legal’ if it satisfies certain well-formedness conditions. We use Greek capitals: \( \Gamma \), \( \Delta \), as meta-variables over contexts.

**Equivalence of types**

A machine can calculate in type theory, by normalising the (often rather complex) terms that represent objects and types. Normalisation leads to an equivalence relation on types. This equivalence relation is denoted by the symbol ‘\( \equiv_{\beta} \)’. The intuition is, that if one has: \( T =_{\beta} D \) then \( T \) and \( D \) basically stand for the same type. The relation \( =_{\beta} \) is decidable.

**Judgements**

A *judgement* is a relation between a context and a statement. A judgement is written as:

\[ \Gamma \vdash X : T \]

Type theory gives a recursive set of rules that define the set of so-called *valid* judgements. Given these rules, validity of judgements is decidable. If a given judgement \( \Gamma \vdash X : T \) is valid, we can interpret this as follows.

- That \( \Gamma \) is a well-formed representation of an agent’s background knowledge. Technically, \( \Gamma \) is called a *legal* context.
• That $X$ is an inhabitant of $T$, given the background knowledge represented in the context $\Gamma$. To interpret what it means that $X$ is an inhabitant of $T$, there are two cases that are of interest to us now:

  - The case where $X$ is a type. In this case the term $T$ must be the kind $\ast$, and the judgement expresses that $X$ is a concept of the agent with the background knowledge represented by the context $\Gamma$.

  - The case where $X$ is an object. In this case the term $T$ must be a type, and the judgement expresses that $X$ is an instance of the concept represented by the type $T$ for an agent with background knowledge represented by the context $\Gamma$.

**Formalising knowledge**

Given the above interpretation of judgements, we can now answer the questions on page 15.

1. How to represent the background knowledge of an agent?
   The background knowledge of an agent is represented by a legal context. A context is legal if it occurs in a judgement, i.e. iff there exist expressions $X$ and $T$ such that: $\Gamma \vdash X : T$.

2. Which complex concepts are meaningful, given the atomic concepts in the background knowledge of the agent?
   A concept $T$ is meaningful given a background $\Gamma$ iff: $\Gamma \vdash T : \ast$.

3. How can the agent decide whether concepts are equal?
   Two concepts $T$ and $D$ are equal if $T =_\beta D$.

4. How can the background knowledge of the agent be extended with the result of observations?
   If $T$ is a concept in $\Gamma$, and it is observable, and an instance of it has been successfully observed, then one can extend $\Gamma$. This extension leads to a new context $\Gamma, x : T$ (where $x$ is a fresh variable). The resulting context will again be a legal one. Note: on the extended context new concepts may be formed, and new conclusions may be drawn.

5. How can the agent draw conclusions within its actual background knowledge?
   If the judgements $\Gamma \vdash T : \ast$ and $\Gamma \vdash X : T$ are valid, then the agent may conclude that $X$ is an instance of the concept $T$ whose existence follows from the knowledge in $\Gamma$. 
1.5. KNOWLEDGE AND REALITY

In this way we have answered all the questions from page 15, except the last one. Which of the concepts that are meaningful in a knowledge state are (directly) observable? This question will be answered in chapter 3. It is a complex question, as it depends on the observational abilities of the agent, abilities that can vary from agent to to agent.

1.5.4 Intensionality of concepts

What is essential to the model that we present here is that the connection between the knowledge state of the agent and the surrounding reality passes entirely through the observational concepts of the agent. Such concepts relate to the world in terms of the agent’s observational abilities, i.e. through computational criteria that the agent can apply to the external world.

Note that there can be many concepts in the background knowledge of the agent that are never directly observable. These concepts can only be connected to the observable ones through ‘mental constructions’ in the background knowledge of the agent. The agent may, within the context \( \Gamma \) that represents his background knowledge, construct instances of such unobservable concepts that are rooted in observations of instances from other concepts.

For a given agent, there are consequently two ways to find an instance of a certain concept \( T \).

- If the type \( T \) is observable, the agent may observe an instance directly.
- Otherwise, the agent may be successful in constructing an instance of \( T \) within its background knowledge, possibly by first extending this knowledge with observed instances of other observable types.

Note that the observable types are not necessarily the same for all agents. For example, an agent which cannot see colours, cannot see that a certain light is red. However, this agent can still infer that the light is red. This may follow from background knowledge when combined with further observational facts. The light may happen to be a traffic light, for instance.

We see that in this computational setting the relation between knowledge and reality no longer rests on a mapping from constants in some logic to the different ‘individual entities’ that exist in the real world. The knowledge of an agent is related to reality on the basis of observations by the agent, through the correspondence between the concepts in the knowledge state with observational criteria. The knowledge of the agent does not reflect what the world ‘really’ is like, but reflects the observationally based distinctions that the agent is able to make, given the way in which he perceives the world.

In his judgements the agent can now distinguish between concepts \( T \) and \( D \) if they are not equal under the equivalence relation \( =_\beta \). For this reason the resulting distinctions are very fine-grained ones. If two concepts \( T \) and \( D \) are
CHAPTER 1. INTRODUCTION

non-distinguishable, i.e. if \( T =_B D \) then they involve similar computations, they apply the same computational criterion to the external world. Thus, in a world where \( T \) is inhabited, \( D \) will also be inhabited.

However, the converse does not hold. It is not the case that if the computational criteria \( T \) and \( D \) always yield the same result, that it follows that \( T =_B D \). This cannot be, if only for the simple reason that the relation \( =_B \) is decidable, and the question whether two computations always yield the same result is undecidable. So, the equivalence relation \( =_B \) that holds between concepts is also more fine-grained than logical equivalence.

It should come as no surprise, therefore, that the model we have sketched here is equipped rather well to deal with the problems of intensionality, as we will see in chapter 3.

1.6 Reasoning about time

As the external world changes, it is not enough to develop a model of an agent that collects knowledge about a static world. We have to understand how that agent is able to reason about the changes that are taking place, and how he is able to adapt this knowledge to take these changes into account. There are many different formal methods that have been proposed to deal with change and time. Before we explain our own approach, we will briefly discuss some of these here, making a crude distinction between ‘model-oriented’ and ‘logical’ methods.

1.6.1 Model-oriented methods

The rationale behind most of these methods is the desire to verify the correctness of concurrent systems.

Some methods that have been developed are algebraic, like Milner’s Calculus of Communicating Systems [Milner, 1989] and various Process Algebras [Baeten & Weijland, 1990], [Baeten & Bergstra, 1991]. These algebras are used to study the interaction between different systems. A system that exhibits a certain type of behaviour is represented by an algebraic term that ‘mimics’ this behaviour. Given a correspondence between input and output actions for two systems, the terms corresponding to these systems can be combined (by so-called parallel composition) to model the behaviour of the resulting system. Inspection of the resulting term allows us to investigate the resulting behaviour. In this way one can detect deadlock, or verify that the resulting system satisfies certain desired properties, etc. Practically, this method can be used to guarantee that programs or chips interact in intended ways. The method has the advantage that it can be mechanised quite well. A problem is that the terms involved can become extremely large.
Another popular method is 'model checking'. The idea is to verify the properties of systems by checking all the states in which the systems may possibly be found against temporal logic specifications, see [Clarke & al., 1986], [Clarke & al., 1994]. Though this method (also) tends to suffer from state explosion, it has already been successfully applied to commercial problems. Using smart abstraction techniques it is nowadays feasible to use this method for state spaces with sizes of up to \(10^{100}\) states [Clarke & al., 1993].

Both the algebraic and the model checking approach are very useful. However, they are designed to support the understanding of systems that we create ourselves, like computer chips and concurrent programs. Accordingly, the methods assume that one has exhaustive knowledge about the systems that one wants to reason about.

For this reason, these methods, (which as a rule do not support quantification over objects) are not suited for agents that have to reason about an external reality. Real world agents routinely have to reason about a world that contains an unknown number of objects, and must also deal with objects whose behaviour is only known to a limited extent. To reason about such systems, we need logical methods.

### 1.6.2 Logical methods

It is well known that it is quite hard to give a logical formalisation of a changing world. If one is not very careful, facts about the world at different moments can easily contradict each other.

In the literature, this problem is usually addressed by a well-known manœuvre: one introduces 'situations' [McCarthy & Hayes, 1969] or 'modalities' [Prior, 1967], or temporal intervals, and qualifies propositions by adding temporal information to them. There are essentially three ways to do this:

- One uses a modal operator, that can get its meaning in terms of Kripke Frames. Working along this line leads to modal logics as described in, for instance [Prior, 1967], [Benthem, 1983].

- One annotates propositions with an extra argument that refers to the situations in which they hold. In this way one obtains the situational calculus [McCarthy & Hayes, 1969]. A popular (simplified) variant of the situational calculus is the 'STEPS'-format [Fikes & Nilsson].

- One adds to predicates an extra argument referring to a time element to make room for the necessary distinctions. An important representative of this approach is Allen's 'temporal logic' [Allen, 1983], [Allen, 1984], and [Allen & Hayes, 1989]. This is a logic with reified events.

The use of modal operators, of a situation argument or of an extra time argument in all predicates will prevent that propositions about different
times, situations or worlds are directly combined. This works, but only to a
certain extent, as it invokes another problem: the frame problem.

1.6.3 The frame problem

A problem arises because of the need to transport information between dif-
ferent modalities or between different moments in time. In particular, one
often — but not always — wants to assert that something that is true now, is
also true the next moment. A simple way to do this is by adding appropriate
axioms. However, it turns out that the number of axioms needed tends to
grow without limit. This problem is often referred to as the ‘frame problem’
[McCarthy & Hayes, 1969], [Shoham & McDermott 1987]. If one wants to
reason about time, one has to adopt some way to deal with this problem.

From a technical viewpoint there are basically two approaches to the
problem:

- Abandon the use of frame axioms altogether, and employ models that
  In general these approaches are not very robust, and fail on simple ex-
  amples. For a detailed account of the problems see [Sandewall, 1994].

- Use explicit frame axioms, and smart methods to keep their number
  as small as possible. An important trick is that one distinguishes
  kinds of events and specifies for every property what kind of event
  can change it. This technique, called explanation closure [Haas, 1987],
  [Schubert, 1990] can indeed dramatically reduce the number of frame
  axioms needed, as shown in [Schubert, 1990]. The great advantage of
  this approach is that one does not have to leave the confines of predicate
  logic.

For a more elaborate discussion of the many issues involved see for instance,
[Ferguson, 1995] pp. 2-60.

1.6.4 Our approach

The approach that we take in this thesis is new, and rather different from
the ones that we mentioned above.

As we consider the problem of real-world change as too difficult to handle,
we limit ourselves to the understanding of discrete event systems, that are ade-
quate to model changes in cyberspace. In comparison to real-world objects,
discrete systems are simple and easy to understand.

In this restricted setting the problem is easier to analyse. As our analysis
shows, it is not so difficult to axiomatise discrete change and the problems
that arise seem to be due to attempts to axiomatise change in an incomplete ontology. It appears that an approach which tries to directly axiomatise the effects that events have on propositions, aims at the wrong goal.

To understand this, it is instructive to think about the changing states of affairs that go on behind a computer screen. At a convenient level of abstraction, the basic happenings in cyberspace are assignments, i.e. changes of the value of one or more variables in your program's 'memory'. These assignments are always local, especially at the right level of abstraction. In most programs, and in particular in object-oriented ones, the state of a program can be regarded as a tree-like hierarchy, and the changes that are the result of various events are localised within certain leaves or certain branches of that tree.

Despite the fact that the effects of a single event are local, and can be described and executed in fairly simple ways, one single event does still affect (quite literally) countless propositions. The question therefore poses itself: "How can we now determine whether an event affects a certain proposition?" We believe the key to answering this question lies in the observation that propositions are not necessarily non-reducible notions. One can, in many cases, reduce the question whether a proposition holds to a condition on particular structures. In fact, in cyberspace computers can always reduce a time-dependent proposition in this way. For this is what they always do if they have to determine whether a certain condition holds in order to perform a certain action. If we know how propositions can be reduced to conditions on structure, the problem is essentially solved. Combining the information about the way in which propositions can be reduced to structure with the information about the structural effects of events, we can now — in principle — deduce which events will affect what propositions. The challenge is, of course, to work this out technically. To do so, we have to develop an ontology that combines events, objects, the states of objects, propositions, a time order, etc. Interestingly, one of the more basic categories in such an ontology is the 'circumstances of a given object at a given moment'. We can only analyze propositions in terms of conditions on structure — and that is the condition sine qua non of the whole approach — if we consider 'circumstances of a given object at a given moment' as first class citizens. Once this cognitive obstacle has been removed, and the new ontology is in place, one can show that an axiomatisation of this ontology in predicate logic allows us to reason — monotonically — about time, events, and changing properties in cyberspace.

8Alternatively, they may crash your program.
1.7 Communication

The third point that we will address in this thesis is how agents with private knowledge states represented in type theory are able to communicate. Our ultimate aim is to build agents that are able to understand and observe a part of cyberspace and that can communicate with humans. Therefore, we investigate the communication process against the background of the implementation of man-machine interfaces.

1.7.1 The DenK project

In fact, the view on communication that is put forward in this thesis has largely been developed within the context of the DenK project, which aimed at the exploration, formalisation and application of fundamental principles of communication from a computational perspective, in order to support the construction of advanced human-computer interfaces.

The starting point of the DenK-project is the observation that two fundamentally different modes of interaction with computer applications can be distinguished:

- On the one hand, users may interact with an application ‘directly’, in ways that reflect an underlying physical metaphor: objects can be picked up and moved, files thrown into a trash can, etc. Such interaction is, at least in the ideal case, self-evident, and normally does not involve (verbal) language. Interfaces of this kind are called ‘direct manipulation’ interfaces.

- On the other hand, users sometimes interact with an application in a more indirect way using language to express commands or requests. Examples are the use of command line interfaces and query languages, the activation of functions using menus and the use of dialogue boxes. The underlying metaphor is in this case that of a conversation, and interfaces of this kind are called ‘conversational’.

Both types of interactions have their advantages and disadvantages. Though direct interaction is often superior, as it eliminates the ‘referential gap’ between symbolic expressions and their meanings, it often cannot be used. For instance, if one is searching for information, physical metaphors tend to be inappropriate. Also, when a very large number of objects has to be searched or altered, one needs a type of interaction that supports quantification. In such cases, conversational interfaces are far superior.

In the DenK-system both types of interaction are integrated in a way that is natural to the user. The user interacts directly with the application, and is supported by an electronic assistant that is knowledgeable about the
application. While working the application, the user can at the same time consult or instruct the assistant, who also observes the application, and may also interact with it. In this way, the assistant can provide information to the user, or perform simple tasks. The user and the assistant engage in a dialogue where the domain of discourse is the application.

1.7.2 Design of the interface

This idea leads to a design of the interface similar to the one suggested in [Hutchins, 1989], in which the assistant and the user communicate about an application to which they both have access. They can exchange messages, and they can also point at things and observe and manipulate the objects in the domain of discourse. This can be graphically depicted in the DenK 'triangle view' of assisted interaction, as depicted in figure 1.1.

The assistant in the DenK system has a knowledge state that is expressed in type theory. It can observe the domain of discourse and extend its knowledge. It can also perform simple actions inside the domain. To enable the assistant to communicate with the user, it is endowed with language understanding capabilities. It is able to interpret the utterances of the user in the light of its actual knowledge state and react to them.

The implementation of the language understanding capabilities of the assistant is based on the view that dialogue participants use language to perform communicative acts, aimed at changing the addressee's cognitive state in certain ways. This is the general tenet of speech act theory [Austin, 1962], [Searle, 1969] as well as of more recent approaches to dialogue analysis such as Dynamic Interpretation Theory, [Bunt, 1989], [Bunt, 2000], and Communicative Activity Analysis [Allwood & al., 1992]. The utterances of the user
are supposed to signal certain goals, which the assistant tries to understand and to fulfill.

To simplify matters, we assume that control always lies with the user. The assistant does not take the initiative to act or to communicate, but only reacts to the action of the user. This is motivated both by traditional interface design and the fact that it is easier to implement. There are of course situations where an entirely passive attitude is not desirable [Masthoff, 1997].

In DenK (as in general) the interpretation of natural language utterances not only depends on the syntactic structure of the utterance and the meanings of the constituent lexical items, but also on the context of use. This context is represented by the knowledge state of the assistant and by the state of the application domain, as visible to the user and internally observable by the assistant. Language interpretation in the system is designed as a two-stage process:

- The first stage is a context-independent interpretation process, where the semantic consequences of morphosyntactic structure are made explicit. This task is performed by a natural language parser, that creates an underspecified representation of the user’s utterance. This representation leaves open any aspect of meaning that cannot be decided on the basis of syntactic evidence present in the utterance [Verlinden, 1999].

- The second stage is a context-dependent interpretation process where contextual information is taken into account to obtain appropriate interpretations. Content words are related to the concepts of the application domain, pronominal anaphors and definite descriptions are related to the objects the user intend to refer to, etc. This process uses the assistant’s knowledge about the domain, as well as information about recent utterances in the dialogue to interpret the underspecified representations in terms of type-theoretical constructs [Kievit, 1998].

If the semantic content of the utterance can be given a meaningful contextual interpretation, the assistant will update its knowledge state accordingly, and tries to react appropriately. The assistant considers two types of action in response to the user’s communicative behaviour:

- Domain actions, that change the state of the domain.

- Communicative actions, that change the information state of the user.

The choice of action depends on the semantic content and the communicative functions of the user’s utterance, as determined by the context-dependent interpretation process. The assistant uses a system of pragmatic rules to determine appropriate reactions; see [Piwek, 1998].
1.7. COMMUNICATION

1.7.3 Type theory as a semantic framework

In this thesis we will not go into any of the details of natural language processing in the DenK system, but only discuss two questions that arise when natural language utterances are to be interpreted using type theory as a semantic framework.

- The first question is how type theory may be used as a semantic representation language in natural language interfaces. To answer this question, we will compare type theory with Discourse Representation Theory (DRT), a well-established theory of natural language interpretation [Kamp & Reyle, 1993]. We show, using a simplified version of an argument presented in [Ahn & Kolb, 1990], that the semantic representations that are used in DRT, so-called Discourse Representation Structures (DRSs), can be translated into types in type theory in a systematic fashion. Subsequently, we argue that DRT semantics is a limit case of a more general approach, in which utterances are judged from the subjective perspective of an agent with a given knowledge state. As our model of an agent's knowledge (a type-theoretical context) is inherently partial, one can also understand how this knowledge may be extended when an agent encounters new information.

- The second question is how agents that have type-theoretical knowledge states may be able to exchange information, given some common vocabulary. Our treatment is in line with [Ahn & Borghuis, 1999], [Ahn, 1994]. We will consider the situation of two agents that have type-theoretical knowledge states, and try to model the communication process. To do so, we will have to go beyond the simple model of type-theoretical knowledge states that we have used so far, and use knowledge states that distinguish between two kinds of knowledge: knowledge that is private, and knowledge that is common, i.e. believed to be shared with the other discourse participant. We draw attention to an interesting phenomenon: within a communication situation the types that can be constructed in type-theoretical contexts can be used in two different ways. They can be used either to convey new information, thus extending the context of the listener, or as a requirement which the listener has to fill in on the basis of his own knowledge. In this way it is possible to give an account of the direction of information flows in communication. Among other things, this provides interesting insights in the function of definite and indefinite determiners, and also provides a simple model for the relationship between questions and answers.
1.8 Overview of the thesis

The thesis deals with three main topics:

- In chapters 2 and 3 we describe the use of type-theory for modelling the knowledge state of an agent, and we discuss how such a knowledge state is related to an external reality.

- In chapters 4 and 5 we focus on the modelling of time and change, in order to allow the agent to deal with an external reality that is not static.

- In chapters 6 and 7 we will show how type-theoretical knowledge states can be related to an existing linguistic theory, and how the communication process can be modelled between two agents whose knowledge states are represented in type theory.

In chapter 8 we present our conclusions and discuss possible directions of future work. In the remainder of this chapter we now briefly summarise the chapters 2-7.

1.8.1 Summary of chapters 2-7

Chapter 2: Type theory

In chapter 2 we introduce type theory, which is the ‘logical’ foundation that we use to model knowledge states. In the first part of chapter 2 we give a rather informal introduction, aimed at those readers that have had some exposure to lambda-calculus but are not familiar with type theory. In this introduction, which is based on Barendregt’s Pure Type Systems, (PTS) we introduce the notions of ‘context’ and ‘judgement’, and exhibit the relation between type theory and $\lambda$-calculus. We also present an example that shows how type-theoretical contexts are formed, and how judgements can be proved within them. Next, for the sake of completeness and precision, we present a standard formal definition. We then go on to discuss some extensions of standard type systems, the use of contexts with definitions, the so-called ‘books’, as well as sigma types and inheritance.

Subsequently, we informally discuss the meaning of types, and introduce a view on type theory in which types are interpreted as specifications. We go on to consider the problem of constructing inhabitants for given specifications, which gives rise to the notions of ‘extending segment’ and ‘clause’.

Chapter 3: Knowledge and observation

In this chapter it is shown how type-theoretical contexts, or more precisely, books, may be used to represent the information of an agent about an external
1.8. OVERVIEW OF THE THESIS

We can interpret such books as meaningful knowledge states — which are really related to an external reality — by grounding them semantically. We argue that semantic grounding should not be construed as an abstract mathematical relation between a context and an external reality, but that it has to be based on the agent's individual observational abilities. Accordingly, semantic grounding is modelled as the agent's ability to construct inhabitants of concepts on an observational basis. In this way the agent can extend its information state as a result of observations.

To illustrate all this, we develop an example in some detail. Next, we show that the resulting information states have many interesting and useful properties. Apart from being partial, extensible, and computational, they are semantically grounded in a subjective way, and intensional in the sense that they can deal in a meaningful way with non-existent 'objects', and can also distinguish between co-extensional 'objects' like the evening star and the morning star.

Finally, we show that such information states lead rather naturally to a foundational method of belief revision, that has some prospect of being computationally tractable.

Chapter 4: Modelling dynamic objects

In chapter 4 we consider the formal representation of time and change. We propose a new and we believe rather satisfying solution to this problem: giving up the (unwarranted) assumption that the objects of the logic correspond directly to the objects in the world, we succeed in developing an ontology in which both events and objects figure, and which allows us to indicate how changes that are due to certain events are localised within certain objects. The crux of this ontology is that it enables one to distinguish between a historical object, the state of this object at a certain moment in time, and the circumstances of this object at a certain moment in time. Formalising this ontology, it is possible to develop a method that allows us to reason about change in a monotonic framework. This method is illustrated with a few examples.

Chapter 5: Time, types and memory

In this chapter we transfer the ideas of chapter 4 to the kind of knowledge states that we proposed in chapter 3. To do so, we first reinterpret some of the concepts that were introduced in chapter 3 in the light of the more complex ontology presented in chapter 4. In particular, we investigate the meaning of 'objects' and what happens when objects are observed. Next, we show how to to construct a knowledge state in which information about different moments in time can be combined, by incorporating axioms that
describe the ‘structure’ of time. Finally, we investigate how the knowledge state of an agent that is constructed in this way might develop, and what the resulting view of such an agent about the external reality amounts to.

Chapter 6: Type theory and DRT

In this chapter we show how type-theoretical knowledge states can be related to an existing theory of natural language interpretation, Discourse Representation Theory (DRT). We present a translation from the semantic representations of DRT, the so-called DRSs, to type theory, and show how DRT’s truth-conditional semantics can be considered as a limit case of a more fine-grained ‘mentalistic’ semantics based on satisfiability with respect to a given context.

Chapter 7: Inter-agent communication

In chapter 7, we consider the communication process between two agents. We show how information transfer between knowledge states can be achieved. Assuming that types can be related to words, we offer a simple mechanism for information exchange, which also sheds some light on the representation of questions and assertions, and on the function of determiners.
Chapter 2

Type theory

In this thesis we defend the view that it is profitable to use so-called type theoretical contexts to model knowledge states. Given the importance of type theory for the present thesis, we will, in this chapter, present a short and practical introduction into type theory. Familiarity with λ-calculus is assumed. Type theory is a flourishing subject which has branched\(^1\) in several directions, and nowadays there are various systems in use. The introduction given here limits itself to Barendregt’s so-called Pure Type Systems. In section 2.3 this introduction is completed with a standard formal definition.

We go on to discuss some extensions of standard type theory. In section 2.4 we introduce contexts with definitions, the so-called ‘books’. In section 2.5 we discuss so-called Σ-types and inheritance. In section 2.6 we informally discuss the meaning of types, and introduce a view on type theory in which types are interpreted as ‘specifications’. We go on to consider the problem of constructing inhabitants for given specifications, which gives rise to the notions of ‘extending segment’ and ‘clause’.

2.1 What is a type system

When one opens an arbitrary mathematical textbook and studies the mathematical vernacular, one may observe that the text has a particular structure that is characteristic for mathematical texts. First, a number of individuals having certain properties is assumed. Then, it is demonstrated that if such individuals do indeed exist, that they are bound to have certain other properties. Next, one derives further consequences from these first conclusions, and so on. Thus, from a formal viewpoint, mathematical texts mainly

\(^1\)For a discussion of some of the underlying issues, see [Luo, 1994, pp. 8-15]
show which consequences follow from certain assumptions, and how these consequences follow from the assumptions.

As was first demonstrated by De Bruijn [de Bruijn, 1980], mathematical texts can be formalised in certain explicitly typed λ-calculi, the so-called type systems. These allow us to represent the two main ingredients that are found in a mathematical text:

- The body of assumptions.
- The consequences that are drawn from it.

Both the body of assumptions and its consequences are represented by syntactic structures that are set up according to certain rules. These rules assure the following:

- Any conclusion that is drawn is a necessary consequence of the given assumptions.
- Any new assumption that is made, makes sense against the background of the things that have already been assumed.

To illustrate the last point; we can only assume that a number ‘\( p \)’ is a ‘prime’, if we already believe that there are such things as numbers, that ‘\( p \)’ is one of them, and that there is a property of being ‘prime’, that is applicable to numbers\(^2\).

The rules of the type systems are rather uniform, and reflect the (somewhat surprising) observation that there is no formal difference between the way in which traditional mathematical objects – like numbers and functions – are to be combined, and the way in which justifications for propositions – like proofs – are to be combined.

A consequence of this unified view is that the syntactic structures that are combined can represent rather different mathematical objects. They may represent numbers, propositions, proofs, connectives, predicates, functions etc. To harness this plethora of species, some order has to be imposed. This is achieved through the way in which, based on the current body of assumptions, the type system assigns different types to the various syntactic structures. More precisely, given the body of assumptions, the rules of the type system determine:

- Whether and how structures with given types can be combined.
- What the type of a legal composition of typed structures must be.

\(^2\)For linguists: to assume that the king of France is bald, one must already believe that ‘France’ is a country, that countries have rulers, that France is indeed ruled by a king, that kings are men, and that there is a property of ‘baldness’ that is applicable to men.
2.1. WHAT IS A TYPE SYSTEM

- What the type of possible extensions of the current body of assumptions can be.

Essential is that hypothetical reasoning is modelled in type systems as a formal syntactic process. The rules of type systems constrain this process in such a way that it can be interpreted mathematically.

2.1.1 Contexts

As we have stressed before, mathematical reasoning is hypothetical, in the sense that it investigates the consequences of certain assumptions. In such an activity, it is of course imperative to keep track of all the assumptions that one currently entertains. To do this, type systems use a formal representation of the current 'body of assumptions'. Such a representation is called a context.

A context is a listing of everything that has been assumed so far. Formally, it consists of a sequence of variable declarations, in which variables are declared to be inhabitants of certain types. A context is written:

\[ x_1 : T_1, x_2 : T_2, \ldots, x_n : T_n \]

Here the \( T_i \) (the types) are terms that represent the current assumptions and the \( x_i \) are variables that stand for the justifications of these assumptions. The pairs \( x_i : T_i \) consisting of a variable and its type are called declarations.

Contexts are not fixed. It is always possible to make new assumptions, which means that contexts remain 'open', and can always be extended. The ways in which contexts can be extended will be constrained by the rules of the type system. Contexts that are formed in accordance with these rules are called 'legal' contexts. In the sequel, when discussing contexts, we use Greek capitals, in particular \( \Gamma \) and \( \Delta \), as meta-variables that range over contexts, legal or otherwise. The symbol 'e' is used to denote the empty context.

2.1.2 Typing rules

Type systems are explicitly typed lambda-calculi. The syntactic structures that play a role in these calculi are \( \lambda \)-terms. These are application or abstraction expressions that are constructed with the help of the variables that are declared in the context.

Given a particular context, i.e. given an enumeration of variables with their types, the rules of the type system constrain the ways in which these variables can be combined into legal \( \lambda \)-terms, and determine the type of any legal lambda-term. A judgement, like

\[ \Gamma \vdash E : T \]
asserts that a term (‘E’) has a certain type (‘T’), given the assumptions in a context (‘Γ’). Or, in other words, it asserts that the term T has the term E as an inhabitant. The rules of the type system enable us to infer which judgements are valid. The valid judgements also determine which terms and which contexts are legal: terms and contexts are legal if they occur in a valid judgement.

Examples of judgements

A few examples may clarify things. Let’s assume that we have constructed a context Γ in which we have declared a number of well-known mathematical ideas; among them the type ‘nat’ of natural numbers, the number ‘one’, the successor function ‘suc’ and the predicate ‘odd’. How this can be done will be explained shortly. Examples of judgements that one can make given this context, are:

- The number one is an inhabitant of the type of natural numbers:

  \[ \Gamma \vdash one : nat \]

- The successor function is a function from natural numbers to natural numbers:

  \[ \Gamma \vdash suc : (nat \to nat) \]

- The successor of one is a natural number:

  \[ \Gamma \vdash (suc\ one) : nat \]

- The composition of the successor function with itself yields a function from natural numbers to natural numbers:

  \[ \Gamma \vdash (\lambda x : nat.(suc(suc\ x))) : (nat \to nat) \]

The above examples are all self-evident. They assert that familiar mathematical objects are inhabitants of certain standard types. However, the inhabitant relation ‘;’ is hierarchical. The standard types themselves can also appear as inhabitants in other more abstract judgements. It is customary to use the the symbol ‘*’ for the type of all standard types. For instance, one has the following valid judgements:

- The type of natural numbers is a (standard) type:

  \[ \Gamma \vdash nat : * \]
2.1. WHAT IS A TYPE SYSTEM

- The type of functions from natural numbers to natural numbers is a (standard) type:

\[ \Gamma \vdash (\text{nat} \rightarrow \text{nat}) : * \]

Surprisingly, perhaps, propositions are also standard types. This will be explained later.

- The proposition that ‘one’ is ‘odd’ is a type:

\[ \Gamma \vdash (\text{odd }\text{one}) : * \]

- The (false) proposition that the successor of ‘one’ is ‘odd’ is also a type:

\[ \Gamma \vdash (\text{odd}(\text{suc }\text{one})) : * \]

Apart from standard types, there can also be functions that deliver standard types. These are called constructors. Examples are:

- The identity function over types is a constructor:

\[ \Gamma \vdash (\lambda t : *.t) : (* \rightarrow *) \]

- The predicate ‘odd’ is also a constructor:

\[ \Gamma \vdash \text{odd} : (\text{nat} \rightarrow *) \]

Finally, there are the expressions like * and (\text{nat} \rightarrow *), etc. These are called ‘kinds’. They can also occur in judgements. The symbol ‘\[\square\]’ is used to denote the type of all kinds. Thus:

- The type of all standard types is a kind:

\[ \Gamma \vdash * : \square \]

- The type of all predicates over natural numbers is a kind:

\[ \Gamma \vdash (\text{nat} \rightarrow *) : \square \]

2.1.3 Terms

The precise syntax of legal terms is complicated by the fact that these terms have to obey type restrictions, which makes this syntax dependent upon the valid judgements. As indicated, the set of all legal terms is defined as the set of all terms that can occur in valid judgements. Loosely speaking, the terms that occur in the type systems are either:

- variables that are declared in the context.

- \(\lambda\)-expressions in an explicitly typed \(\lambda\)-calculus.

- arrow-expressions, like: \(A \rightarrow B\).

- sorts: the terms \(\square\) and * are called ‘sorts’.
2.1.4 Levels of the hierarchy

Type theories come in different varieties, but most have at least two sorts. Typically, these sorts are written as: ‘□’ and ‘∗’. In all type systems that we consider, the type-hierarchy has 4 levels. The top of this hierarchy is formed by the sort □. The sort ∗ is one of the inhabitants of □, so, as figure 2.1 indicates, one has:

\[ ∗ : □ \]

The terms that are found at the three lower levels of the hierarchy are distinguished by different names. The level directly below the top of the hierarchy is formed by the so-called kinds which are inhabitants of □. In particular, ∗ is a kind. Other examples of kinds are: \( (∗ → ∗) \), \( (nat → ∗) \) etc. All kinds can have inhabitants. The inhabitants of kinds are the so-called constructors. Examples of constructors are: predicates, propositions, and, as will become apparent, connectives. As ∗, which is a sort, is a particularly important kind, all the constructors that are inhabitants of ∗ are given a special designation and are often – rather confusingly perhaps – designated as types. These (standard) types are the only constructors that can have inhabitants. Propositions, function types and standard types are inhabitants of ∗, so they can have inhabitants. On the lowest level, we find the so-called objects, that are inhabitants of these types. The objects are at the bottom of the hierarchy and cannot have inhabitants.
2.1. WHAT IS A TYPE SYSTEM

2.1.5 Term construction

Judgements can involve rather complex terms. These terms are constructed by combining simpler ones. The ways in which terms can be combined are governed by the principles of a λ-calculus. The λ-calculus in question is explicitly typed. In the simpler type systems the types of the λ-terms are either atomic types (typically 'nat' 'boolean', etc.) or are arrow expressions, like: (nat → nat). In such a calculus, every term has a type, and this type constrains the way in which it can be combined with other terms. The basic principle that guides the combination of λ-terms is extremely simple: λ-terms can be combined using application or abstraction. Functions have a domain which is a type and a range which is also a type. A function \( f \) with domain \( A \) and range \( B \) has the type \( (A \rightarrow B) \). This function can only be applied to a term that inhabits the domain of the function, the type \( A \). If the function \( f \) is applied to an inhabitant of \( A \) the resulting application is a term which inhabits \( B \), the range of the function.

2.1.6 Some simple examples

Though we have not given any formal rules yet, a simple intuitive example may clarify how judgements are related to term construction in the λ-calculus. Consider a context \( \Gamma \) in which we assume the existence of the types \( A \) and \( B \) – these are declared to be inhabitants of the kind \( * \) – an object \( a \) that is an inhabitant of the type \( A \) and an object \( f \) which is an inhabitant of the function-type \( (A \rightarrow B) \). Formally this context \( \Gamma \) reads as follows:

\[
A : *, B : *, a : A, f : (A \rightarrow B)
\]

In this context, the objects \( f \) and \( a \) can be combined, and the resulting combination has the type \( B \). This fact is expressed in the following valid judgement:

\[
A : *, B : *, a : A, f : (A \rightarrow B) \vdash (fa) : B
\]

Though the reader cannot formally check this judgement, as no rules have yet been given, the underlying idea will be familiar from the λ-calculus.

As in the λ-calculus, terms cannot only be formed by function-application, but also by λ-abstraction. To abstract over a certain variable in a type system, one has to explicitly specify the type of this variable. For instance, in any context \( \Gamma \) that contains a declaration of a type \( A \) one can construct the legal term \( \lambda x : A . x \) which stands for the identity function on the type \( A \) and will be an inhabitant of the (legal) function-type: \( (A \rightarrow A) \). Thus we have the following valid judgement:

\[
A : * \vdash (\lambda x : A . x) : (A \rightarrow A)
\]
2.1.7 Propositions as types

At first sight there might seem to be an unbridgeable gap between the judgements of a type system and the proofs in a logical system. The type systems we have presented here concern themselves with mathematical constructions and the classification of these constructions, whereas the essential notions in a logical system are axioms, proofs and propositions. The key to bridging this apparent gap is the propositions as types interpretation (cf. [Curry & Feys, 1958]). This interpretation rests on the following observations:

- Large proofs are constructed from smaller ones.

- The way in which (sub)proofs can be combined to form larger proofs, depends in principle only on the propositions that the various (sub)proofs prove.

- There is a direct correspondence between the structure of natural deduction proofs and the terms of the typed lambda-calculus.

- Not all types are necessarily inhabited; not all propositions can be proved.

These observations suggest that, in a type system, it may be possible to regard proofs as individuals. A proof is then taken to be the inhabitant of a type that represents the proposition that it proves. The proposition itself will be an inhabitant of *. In short, propositions can be taken to be types, and proofs can be taken to be objects. If there is a proof for a proposition, then the proposition is inhabited. Propositions may have several proofs, a proposition is the type of all its proofs. Of course, there also are many propositions that do not have proofs. Propositions that have no proofs have no inhabitants, which implies that many types will be empty. Note that this implies that, in type theory, there can be many different empty types. Though the idea to let types represent propositions seems somewhat strange when first encountered, it works extremely well, particularly when formalising natural deduction proofs. In fact, various type systems for λ-calculus do – more or less by accident, it seems, but probably due to some deep mathematical symmetries that are not yet fully understood, as argued in [Girard, 1989] – express rather precisely how proofs are constructed in natural deduction. This works properly because the typing rules constrain the way in which proofs are combined and assure that proofs can only be combined in a ‘logically correct’ way. Thus, within type systems, proofs have explicit formal representations. In fact, they are mathematical individuals that exist on an equal footing with all others, such as numbers or functions.
2.1. WHAT IS A TYPE SYSTEM

Considering the examples from the previous subsection again, it is in-
structive to see that it is indeed possible to interpret the types $A$ and $B$, which are inhabitants of $\star$, as propositions. A context like:

$$A : \star, B : \star, a : A, f : (A \rightarrow B)$$

can be interpreted as containing declarations for the propositions $A$ and $B$, -- these declarations by themselves only introduce the propositions, not proofs of these propositions -- a proof $a$ of the proposition $A$, as well as a proof $f$ for the proposition $(A \rightarrow B)$. The judgement:

$$A : \star, B : \star, a : A, f : (A \rightarrow B) \vdash (fa) : B$$

can be interpreted as showing that by combining the proofs $a$ and $f$ by function-application, we get a proof $(fa)$ for the proposition $B$. As one can see, function application in such cases corresponds to modus ponens.

Abstraction, on the other hand, corresponds to the discharge of assump-
tions in natural deduction proofs; that is why in this same context the term

$$\lambda x : A. x$$

-- which stands for the identity function for type $A$ -- can act as a (trivial) proof of the tautology $(A \rightarrow A)$.

2.1.8 Assumptions in a context

As we have seen, a context is a sequence of assumptions, and can always be extended with new assumptions. Though it may seem that one can always assume anything that one likes, this is not really the case. An assumption must be meaningful -- has to make sense -- given a body of assumptions one tries to extend. In type theory, where a context represents the body of assumptions, the typing rules also restrict the way in which contexts can be extended. These restrictions guarantee that a context can only be extended with a new assumption if such an assumption can be shown to be 'meaningful' in the given context. Formally, it is only legal to extend a context with an -- assumed -- inhabitant of a term $T$, if one can show that the term $T$ already has a type in the current context, and that this type is a sort. As a consequence, the contexts of type systems are extendible, not just with new proofs, but also with new objects or types. Type systems are set up in such a way that all the 'well-formedness constraints' are handled within the system, and are automatically adapted to accommodate new assumptions that are made. This is one of the attractive features which make type systems so versatile. Where conventional logical systems come equipped with a static and externally defined signature -- distinguishing constants, functions and predicates -- the signatures of contexts in type systems are extendible, and are booted from a handful of system constants, the sorts. Subsequently, these contexts can always be extended.
2.2 Example

As all this sounds rather abstract, we will now construct an example context
for one of the simpler type systems, the system known as $\lambda_\omega$. This example
will gradually be extended. The system that we consider in this example
has a number of syntactic rules, given below, which amount to a recursive
definition of valid judgements. These rules are to be interpreted as follows:
A rule

$$\frac{J_1...J_n}{J_0}$$

states that judgement $J_0$ is valid, provided the judgements $J_1...J_n$ are valid.
Using these rules, valid judgements can be constructed recursively. The rules
of $\lambda_\omega$ are given below, where 's' is a meta-variable over sorts, that can take
the values '*' or '□' (the same value throughout the rule) when a rule is used.
Accordingly, rules that contain this variable can be used in two different ways,
depending on the sort that is chosen for 's'.

Definition 1 (Rules of $\lambda_\omega$)

$$\epsilon \vdash * : □ \quad (axiom)$$

The 'axiom' rule ensures that there always are some sorts that have a sort
as type. This is necessary in order to start the construction of a context. It
also states the valid judgement that * has type □ in empty contexts, which
implies that the empty context is legal.

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \quad x \text{ fresh (start)}$$

The 'start' rule allows us to extend a given (legal) context. The constraint
that $A$ must be a sort, as expressed by: $\Gamma \vdash A : s$ ensures that only meaningful
extensions are possible. Of course, $x$ must be a fresh variable, i.e. a variable
not yet used in the context.

$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \quad x \text{ fresh (weakening)}$$

The 'weakening' rule guarantees that context extensions are monotonic, i.e.
that a statement which is derivable in a given context, is derivable in every
legal extension of that context. Of course, $x$ must be a fresh variable.

$$\frac{\Gamma \vdash A : * \quad \Gamma \vdash B : s}{\Gamma \vdash (A \rightarrow B) : s} \quad (arrow)$$
2.2. EXAMPLE

The ‘arrow’ rule regulates the formation of function types.

\[
\Gamma \vdash F : (A \rightarrow B) \quad \Gamma \vdash a : A \\
\frac{\Gamma \vdash (Fa) : B}{(\text{apply})}
\]

The ‘apply’ rule states that the type of an application expression is the range-type of the function, if the argument can be shown to ‘fit’ the function.

\[
\Gamma, x : A \vdash b : B \quad \Gamma \vdash (A \rightarrow B) : s \\
\frac{\Gamma \vdash (\lambda x : A. b) : (A \rightarrow B)}{(\text{abstract})}
\]

The ‘abstraction’ rule allows the creation of functions as \(\lambda\)-expressions.

Given these rules, we can build a legal context of assumptions. A context is legal if it occurs in a valid judgement. In particular, to show that a context \(\Gamma\) is legal, it is sufficient to show that the judgement \(\Gamma \vdash * : \Box\) is valid.

Once a legal context \(\Gamma\) is constructed, interesting judgements that involve this context can be proved. This is facilitated by the following apparently trivial but useful properties of legal contexts:

**Fact 1** Let \(\Gamma, \Gamma \equiv x_1 : T_1, x_2 : T_2 \ldots x_n : T_n, (n \geq 0)\) be a legal context, then:

1. For any index \(i, (i \leq n)\) the judgement \(\Gamma \vdash x_i : T_i\) is valid. I.e. one can select an arbitrary declaration from a valid context to form a valid judgement. *Proof: by induction on the structure of derivations. See [Barendregt, 1992], lemma 5.2.9.*

2. For any index \(i, (i \leq n)\) there is some sort \(s\) such that the judgement \(\Gamma \vdash T_i : s\) is valid. I.e. all the types that occur in the declarations of a valid context inhabit some sort. *Proof: The ‘start’ and ‘weakening’ rules are the only rules that allow us to extend a context with new declarations. They both impose the constraint that the type that occurs in the declaration inhabits some sort.*

Keeping these facts in mind, the usual procedure is to first construct a legal context, and then to proceed to show the validity of certain judgements that involve this context. Later, the legal context can be extended, new judgements can be proved in the extended context, etc. Note that a judgement that is valid on a given context is always valid on a legal extension of this context, due to the weakening rule. In the sequel we write: \(\Gamma \vdash \text{ok}\) when we wish to indicate that the context \(\Gamma\) is legal.
2.2.1 Constructing a legal context

The legal context that we construct will be a rather trivial one, that introduces integers, some functions to construct integers, and a few predicates over integers. This context is constructed making successive valid extensions with the ‘start’ rule, starting from the empty context. These contexts are legal as they appear in some judgement. As the empty context occurs in the judgement

$$\epsilon \vdash * : \Box$$

which is valid due to the ‘axiom’ rule, it is legal. Next we extend this context with a \textit{type} i.e. an inhabitant of ‘*’ that we will interpret as representing the natural numbers. We use the ‘start’ rule with \((s = \Box, A = *)\) to form a larger context. As a variable name for the fresh ‘x’ needed in this rule we choose the symbol ‘nat’. We get the following judgement\(^3\):

$$nat : * \vdash nat : *$$

This judgement show that the context ‘nat : *’ is legal. To indicate this, we write:

$$nat : * \vdash \text{ok}$$

Next, we can extend this context with a declaration of a natural number, like ‘one’. We again use the start rule, \((A = nat, s = *)\) and the fact that \(nat : * \vdash nat : *\) because ‘nat : *’ is a valid context. We get the following judgement:

$$nat : *, one : nat \vdash one : nat$$

So this context is also legal. Next, we want to introduce a notion of a successor function. This is a function from natural numbers to natural numbers. Using the ‘arrow’ rule, it follows that:

$$nat : *, one : nat \vdash (nat \rightarrow nat) : *$$

Using the ‘start’ rule again, as above, we can construct the legal context:

$$nat : *, one : nat, suc : (nat \rightarrow nat)$$

In this context, many judgements are valid, for instance:

$$nat : *, one : nat, suc : (nat \rightarrow nat) \vdash (suc(suc one)) : nat$$

\(^3\)If we had used the rule ‘weakening’ to extend the context, we would have obtained the judgement \(nat : * \vdash * : \Box\). This illustrates the fact that valid contexts can appear in many different judgements, as we noted earlier.
2.2. **EXAMPLE**

Subsequently, we can introduce a predicate over natural numbers. As an example, take the predicate of being an odd number. Such a predicate, when combined with a number, yields a proposition. As propositions are inhabitants of $\ast$, as we have seen in paragraph 2.1.7, the predicate: ‘odd’ must be introduced as an inhabitant of $(\text{nat} \rightarrow \ast)$. As follows from the ‘arrow’ rule, this term is itself a kind, it is an inhabitant of $\Box$. $\Box$ is a sort, so we are indeed allowed to extend our context with a declaration of an inhabitant of $(\text{nat} \rightarrow \ast)$. The new context is legal:

$$\text{nat} : \ast, \text{one} : \text{nat}, \text{suc} : (\text{nat} \rightarrow \text{nat}), \text{odd} : (\text{nat} \rightarrow \ast) \vdash \text{ok} \quad (2.6)$$

In this context, which we abbreviate as ‘$\Gamma_4$’ many judgements are valid, for instance:

$$\Gamma_4 \vdash (\text{odd one}) : \ast$$

And also:

$$\Gamma_4 \vdash (\text{odd (suc one)}) : \ast$$

These judgements show that the terms $(\text{odd one})$ and $(\text{odd (suc one)})$ are inhabitants of $\ast$. They show that these terms are propositions. They do not show that these propositions are true. In fact, the context still does not contain any information that allows us to determine which numbers are odd. To assert that a proposition like: $(\text{odd one})$ is true, it needs a proof, i.e. an inhabitant. If we do assume the existence of such a proof, called ‘$p$’ for instance, we can extend $\Gamma_4$ to the following legal context:

$$\Gamma_4, p : (\text{odd one}) \vdash \text{ok}$$

In this context, which we abbreviate as ‘$\Gamma_5$’, we have the valid judgement:

$$\Gamma_5 \vdash p : (\text{odd one})$$

which expresses that we now indeed have a proof that the number one is an odd number.

All this is very nice, but it does not show how one can construct interesting proofs. To do so, we need the possibility to use universal quantifiers. The system that we have investigated so far does not support this. To express quantification, a somewhat stronger system is needed. Such a system must support the notion of a dependent function type.

2.2.2 **Dependent function types**

If $A$ and $B$ are types, the ‘arrow’ rule allows one to form the function type $(A \rightarrow B)$. In a more powerful system that allows the construction of dependent function types, one can form the term ‘$(\Pi x : A. B)$’, which also denotes
a type, the dependent product of $B$ indexed over $A$. The intuition behind a dependent product is that it is a generalisation of the conventional function type $(A \rightarrow B)$. The point of this generalisation is that it allows the range type of a function to be dependent on the argument of the function\(^4\).

We will see below that such types allow us to express (universal) quantification. Syntactically, the most important aspect of the generalised expression ‘$(\Pi x: A. B)$’ is that the ‘$\Pi$’ acts as a binder for the variable ‘$x$’, which allows the argument $x$ to occur in the range type $B$. In this way the dependency of the range on the argument can be formally expressed. In cases where $x$ does not occur in $B$ it is indeed preferable to write: ‘$(A \rightarrow B)$’ instead of ‘$(\Pi x: A. B)$’.

If one can form dependent function types the ‘arrow’ rule becomes just a special case, and is replaced by a more general rule to form dependent products:

\[
\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash B : s}{\Gamma \vdash (\Pi x: A. B) : s} \quad \text{(product)}
\]

Note that an important difference between the ‘arrow’ and the ‘product’ rule is that that in the latter rule $B$ must be a sort in the context $\Gamma$ extended with a declaration: ‘$x : A$’ of the variable $x$. This circumstance allows $B$ to become dependent on $x$.

The ‘abstraction’ and ‘application’ rules – that also involve an arrow – have to be adapted: For the abstraction rule, this is straightforward:

\[
\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x: A. B) : s}{\Gamma \vdash (\lambda x: A.b) : (\Pi x: A. B)} \quad \text{(abstract)}
\]

For the application rule, the fact that the term $B$ can depend on the argument $x$ means that the appropriate substitutions have to be made when an inhabitant $F$ of a dependent function type is applied to a particular inhabitant $a$ of type $A$. Writing ‘$B[a/x]$’ for the result of substituting the term $a$ for each free occurrence of $x$ in the term $B$, the rule now becomes:

\[
\frac{\Gamma \vdash F : (\Pi x: A. B) \quad \Gamma \vdash a : A}{\Gamma \vdash (Fa) : B[a/x]} \quad \text{(apply)}
\]

The principal use of dependent function types is in the modelling of propositions that contain universal quantifiers. As an example, consider the function $f$ that takes an arbitrary natural number $x$ as an argument, and produces a proof that this number $x$ has a certain property $p$. I.e for any given $x$ the range type of the function is $(p\ x)$. The range of this function

\(^4\)To remember the notation, it helps to note that $(A \rightarrow B)$ is sometimes informally written as: $B^A$ i.e. $B \times B \cdots \times B$ ($A$ times) which, if the $B$’s depend on some $x : A$ generalises to an indexed product, i.e. a ‘$\Pi$’ expression.
2.2. Example

depends on its argument \( x \), and the type of \( f \) cannot be represented in the conventional arrow notation. It is written as: \( \texttt{(\Pi x: \text{nat}. (\text{odd}\ x))} \). Thus the type
\( (\Pi x: \text{nat}. (\text{odd}\ x)) \), is the type of a function that produces for every natural number a proof that the number in question is odd. But while this type exists, it has no inhabitant, as there is no such function. The simple fact that there are numbers for which there is no proof that they are odd – because they are even – implies that there is no way to construct such a function given a reasonable body of assumptions that we could make about numbers. This means that, in every consistent context, this type will be empty.

2.2.3 Encoding quantification

Using dependent function types, we are able to express propositions that involve quantifiers. As we have seen, the type \( (\Pi x: \text{nat}. (\text{odd}\ x)) \) corresponds to the proposition: ‘All integers are odd’. This is not a true proposition, so it will not have an inhabitant. In order to construct an example involving quantification, we need a true proposition that contains a quantifier. For instance: if a number \( x \) is odd, it successor’s successor is also odd. This proposition corresponds to the following type:

\[
(\Pi x: \text{nat}. (\Pi y: (\text{odd}\ y) . (\text{odd}\ (\text{suc}\ (\text{suc}\ x)))))
\]

We are now almost ready to construct a simple proof. Consider the context \( \Gamma_5 \):

\[
nat : *, \text{one} : \text{nat}, \text{suc} : (\text{nat} \rightarrow \text{nat}), odd : (\text{nat} \rightarrow *), p : (\text{odd one}) \vdash \text{ok}
\]

To extend this context with a proof that the successor of the successor of an odd number is odd, we declare a new variable ‘\( ax_2 \)’ which is an inhabitant of the type associated to this proposition, i.e.:

\[
ax_2 : (\Pi x: \text{nat}. (\Pi y: (\text{odd}\ y) . (\text{odd}\ (\text{suc}\ (\text{suc}\ x)))))
\]

The resulting context, that we call ‘\( \Gamma_6 \)’, is indeed a legal context:

\[
\Gamma_5, ax_2 : (\Pi x: \text{nat}. (\Pi y: (\text{odd}\ y) . (\text{odd}\ (\text{suc}\ (\text{suc}\ x))))) \vdash \text{ok}
\]

because one can prove that the type of \( ax_2 \) is an inhabitant of \( * \). Within the context \( \Gamma_6 \), the following judgement is valid:

\[
\Gamma_6 \vdash ((ax_2 \ \text{one}) \ p) : (\text{odd}\ (\text{suc}\ (\text{suc}\ \text{one})))
\]

This judgement expresses that \( ((ax_2 \ \text{one}) \ p) \) is a proof of the proposition \( (\text{odd}\ (\text{suc}\ (\text{suc}\ \text{one}))) \). More elaborate proofs for higher numbers can also be constructed.
2.2.4 Checking proofs

As the reader can see from the example, type systems make formal representation of mathematical proofs possible. Because the rules of the type system are strictly formal, and have certain pleasant (normalisation) properties, it is actually decidable whether a given judgement is valid. This means that the validity of judgements can be checked by a machine. Thus, when given the judgement:

$$\Gamma_6 \vdash ((ax_2 \ one)p) : (odd \ (suc(suc\ one)))$$

a machine is able to establish that, given the assumptions in $\Gamma_6$, the object: $((ax_2 \ one)p)$ is a proof of the proposition: $(odd \ (suc(suc\ one)))$ in $\Gamma_6$.

This is not only true in theory, and for small and trivial examples like the one given here, but also in practice, and for far larger examples. As a matter of fact, the contents of entire mathematical textbooks can be formally expressed, and consequently be checked by a machine. This was first demonstrated on a large scale in the pioneering work of Van Bentham Jutting, who translated the whole of Landau’s “Grundlagen” (a work of 173 pages) in a type-theoretical language called AUTOMATH [de Bruijn, 1980], thereby enabling the computer to check the text, see [Van Bentham Jutting, 1977].

2.3 Pure Type Systems

The AUTOMATH language was an early precursor of modern type systems. In the past 30 years, various type systems have been proposed. The expressive power of these systems is largely determined by the complexity of the types that they allow to be formed. In the early 90’s Barendregt pointed out that many of the existing type systems were all members of one single family, and showed that these systems can be characterised by the same set of rules, provided this set is appropriately parametrised. See [Barendregt& Hémerik, 1990],[Barendregt, 1991],[Barendregt, 1992]. Members of this family only differ with respect to one particular rule, the rule for: ‘product formation’ (cf. ‘product’ rule on page 44). This rule determines over which kinds of individuals one is allowed to abstract. The systems within this family are nowadays known as ‘Pure Type Systems’ (PTS). One of these systems will be used extensively throughout this book.

It is customary not to present a particular type system on its own, but as a certain member of the family of PTS. We will follow this tradition, and first present the general definition of Pure Type Systems, following the notation and conventions of Barendregt [Barendregt, 1992].

---

5It turns out that the entire work contained only one minor error.
6For an account of the historical and mathematical development of type theory, see [Laan, 1997].
2.3. PURE TYPE SYSTEMS

2.3.1 Syntactic structures

The definition of PTS is relative to a set of ‘sorts’ and a set of variables:

Introduction 1 Let $S$ be a specific non-empty set of constants.

The members of $S$ are called sorts. Typically, $S = \{\Box, \ast\}$

Introduction 2 Let $V$ be some countably infinite set of variables.

Typically, the set $V$ contains alphanumerical strings.

Definition 2 (Pseudo-term) Given these sets, the set $T$ of pseudo-terms is defined as follows: $T ::= S|V|(\Pi V : T.T)|(\lambda V : T.T)|(TT)$

The set of pseudo-terms is a context-independently definable set of ‘general’ terms from which the set of ‘legal’ terms that will occur in valid judgements can be picked out later. Note that both ‘$\Pi$’ and ‘$\lambda$’ are variable binders in the usual way. This means that for pseudo-terms the conventional notions of scope and bound or free variables apply. In particular, pseudo-terms are considered to be equal under $\alpha$-conversion i.e. pseudo-terms that only differ in the names of locally bound variables cannot be distinguished.

Definition 3 (Statement) A statement is an expression of the form ‘$A : B$’ with $A, B \in T$. The pseudo-term ‘$A$’ is called the subject of the statement, ‘$B$’ its type.

Definition 4 (Declaration) A declaration is a statement ‘$x : B$’, where ‘$x$’ is a variable, and ‘$B$’ a pseudo-term.

Definition 5 (Pseudo-context) A pseudo-context is a finite sequence of declarations: $x_1 : A_1, \ldots, x_n : A_n$, where all the $x_i$ are distinct.

The set of pseudo-contexts is a syntactically characterised set of ‘general’ contexts, from which the legal ones can be selected later. As before, we use $\Gamma, \Delta$ as meta-variables over (pseudo) contexts.

Definition 6 (Judgement) A judgement written: $\Gamma \vdash E : T$ asserts that a certain statement ‘$E : T$’ is derivable in a context $\Gamma$.

Type systems consist of a set of rules that determine which judgements are valid. These rules are discussed in section 2.3.2.
Reduction relations on terms

Type systems are λ-calculi, and the familiar notions of ‘one-step’ β-reduction, general β-reduction, and β-equality are also defined over pseudo-terms:

- One-step β-reduction is the usual computation rule for λ-expressions:
  \[(\lambda x:A.M)P \rightarrow_\beta M[P/x]\]

- The reduction relation \(\rightarrow_\beta\) on expressions \(A\) and \(B\) is defined as follows
  \(A \rightarrow_\beta B\), iff \(A\) can be reduced to \(B\) using any number of β-reduction steps, so it is the transitive reflexive closure of one-step β-reduction.

- The binary predicate \(=_\beta\) on pseudo-terms is defined as the symmetric transitive closure of \(\rightarrow_\beta\).

2.3.2 The rules

Pure Type Systems are defined through a number of rules which are parametrised with respect to the set of sorts \(S\), the set of variables \(V\), and two other sets, a set \(A\) of ‘axioms’ and a set \(R\) of ‘rules’:

**Introduction 3** Let \(A \subseteq (S \times S)\).

The set of axioms determines which sorts are inhabitants of other sorts, typically, \(A = \{*:\Box\}\).

**Introduction 4** Let \(R \subseteq (S \times S \times S)\).

The set of rules determines over which kind of individuals we are allowed to abstract in the type system at hand. This set of rules determines the expressive power of the system. Given sets \(R\) and \(S\), the inference rules for judgements are given as follows.

**Definition 7** (Recursive definition of \(\Gamma \vdash A : B\))

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\epsilon \vdash s_1 : s_2) (if (s_1 : s_2 \in A)) axiom</td>
<td>The ‘axiom’ rule ensures that there always are some sorts that have a sort as type. This is necessary in order to start the construction of a context. It allows us to derive that ‘*’ has type ‘(\Box)’ in empty contexts.</td>
</tr>
<tr>
<td>(\Gamma \vdash A : s)</td>
<td>((x) fresh) start</td>
</tr>
</tbody>
</table>

\(\Gamma, x : A \vdash x : A\)
2.3. **PURE TYPE SYSTEMS**

The ‘start’ rule allows us to extend a given (legal) context. The constraint that ‘A’ must be a sort, as expressed by: \( \Gamma \vdash A : s \) ensures that only meaningful extensions are possible. Of course, \( x \) must be a fresh variable.

\[
\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \quad (x \text{ fresh}) \quad \text{weakening}
\]

The ‘weakening’ rule guarantees that context extensions are monotonic, i.e. that a statement which is derivable in a given context, is derivable in every legal extension of that context. Of course, \( x \) must be a fresh variable.

\[
\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x : A. B) : s_3} \quad (s_1, s_2, s_3 \in \mathcal{R}) \quad \text{product}
\]

The ‘product’ rule is the most important rule in which PTS can differ. It regulates which kinds of abstractions will be legal. Through the set of ‘rules’ \( \mathcal{R} \) the expressive power of the calculus can be controlled.

\[
\frac{\Gamma \vdash F : (\Pi x : A. B) \quad \Gamma \vdash a : A}{\Gamma \vdash (Fa) : B[a/x]} \quad \text{apply}
\]

In principle, for non-dependent function types, the ‘apply’ rule states the simple fact that the type of an application expression is the range type of the function, if the argument can be shown to ‘fit’ the function. In such a case this rule is related to modus ponens. However, the rule is more complicated due to the possible presence of dependent function types. In the cases where the function type is dependent, one must make the appropriate substitutions. From a logical viewpoint, this is related to the elimination of universal quantifiers.

\[
\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x : A. B) : s}{\Gamma \vdash (\lambda x : A. b) : (\Pi x : A. B)} \quad \text{abstract}
\]

The ‘abstraction’ rule corresponds to the fact that we can form functions by abstraction. In terms of proofs this means that we can make dischargeable assumptions in natural deduction style. As we are required to show that the type of the abstraction produces a legal ‘\( \Pi \)' type, the kinds of dischargeable assumptions one is allowed to make are ultimately controlled by the ‘product’ rule.

\[
\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B \equiv B'}{\Gamma \vdash A : B'} \quad \text{convert}
\]
CHAPTER 2. TYPE THEORY

The ‘convert’ rule expresses that the type of a given expression may be exchanged for any other that is β-equal (as defined in section 2.3.1) provided that the replacing type can be shown to be the inhabitant of a sort.

The rules given above amount to a recursive definition of what is a valid judgement, parametrised with respect to the sets $S$, $A$, and $R$. In the rest of this thesis we will (unless specified otherwise) deal exclusively with the case where:

$$S = \{\Box, *\} \quad \text{and} \quad A = \{* : \Box\}$$

and

$$R = \{(*, *, *), (\Box, *, *), (*, \Box, \Box), (\Box, \Box, \Box)\}$$

The resulting system is often called $\lambda C$, and is also known as $\lambda P\mu$.

2.3.3 Some properties

As discussed in [Coquand, 1985], [Barendregt, 1992] the system $\lambda C$, and all weaker systems can be shown to have a number of pleasant properties.

Decidability

The inference rules for ‘$\vdash$’ given above can be used recursively to determine the type of an object in a given context. This is straightforward because, as the reader may check for himself, the structure of the object whose type is to be calculated always determines which rule to apply. The only rule that could be problematic in this respect is the type conversion rule. It can be proven however, [Barendregt, 1992], Theorem 5.3.33, that the order in which various redexes in a term are reduced is ultimately irrelevant, and that every reduction sequence terminates and yields a unique normal form. The system is said to be strongly normalising. This fact guarantees the decidability of the equality of terms. Consequently, it is also decidable whether judgements in $\lambda C$ are valid.

Unicity of types

One can show that for many PTS, and for $\lambda C$ in particular, the type of a term is unique modulo beta-equality.

---

7In the DenK system, we have not used $\lambda C$, (which is rather strong, it is the strongest one in the Barendregt cube) but a more complicated and slightly weaker system with extra sorts, distinguishing between propositions and sets of objects. This system, called CTT in [Kleivit, 1998] is the strongest system in the ‘Logic Cube’, [Geuvers, 1993], or L-cube, [Barendregt, 1991].
Fact 2 If $\Gamma \vdash A : B$ and $\Gamma \vdash A : B'$ then $B \equiv B'$.

Proof: [Barendregt, 1992], Lemma 5.2.21. This result implies that judgements are rather 'fine-grained', as beta-equality is more specific as logical equivalence.

Subject reduction

On the other hand, $\beta$-reduction within a 'subject' also does not change its type:

Fact 3 If $\Gamma \vdash E : T$ and $E \rightarrow E_1$ then $\Gamma \vdash E_1 : T$

Proof: [Barendregt, 1992], Theorem 5.2.15. This last property ensures, among others, that a proof which can be reduced, will still prove essentially the same proposition after reduction.

Substitutivity

It is possible to show that the system is substitutive:

Fact 4 If we have $\Gamma, x : A, \Delta \vdash M : T$ and $\Gamma \vdash E : A$ then it follows that $\Gamma, (\Delta)[E/x] \vdash (M : T)[E/x]$

Proof: [Barendregt, 1992], Lemma 5.2.11. This property ensures that an assumption 'x : A' which is not needed – because it already follows from earlier assumptions that A is inhabited – can be eliminated from a context, provided the right substitutions are made.

2.4 Books and definitions

Type systems are mainly used by mathematicians and theoretical computer scientists. The systems are used to construct programs, check proofs, or explore foundational issues. Sometimes they are also applied to prove results about the type systems themselves. To support the bookkeeping in such rather complicated reasoning, some extra formal apparatus can be introduced, which helps to bring type systems closer to the mathematical vernacular, and makes them more practical. A very useful formal device, which will also turn out be helpful in dealing with inference in knowledge states, is the introduction of so-called 'definitions' through which contexts are expanded to 'books'.

2.4.1 The use of definitions

Mathematical reasoning is hypothetical, as it reasons on the basis of a number of assumptions. These assumptions determine the generality of the final results. For this reason it is essential that the number of assumptions is kept as small as possible.

In theories that have a small number of axioms, the proofs often involve a large number of reasoning steps. Such proofs are usually not constructed directly from the axioms (i.e. the given assumptions). They are constructed in a step by step process. First, certain lemmas are constructed, that act as tools which can be used over and over again. From these lemmas one builds more interesting lemmas (‘corollaries’), which may finally lead to some overall result. In other words: one does not jump to the end result in ‘one giant leap’ but constructs a path, making one step at the time. This holds not only for the construction of proofs, but also for constructing other mathematical objects like functions, sets or spaces.

It is therefore indispensable to have a mechanism that can be used to store ‘intermediary results’. The mechanism should allow easy access to these results, and enable one to later use them, abstracting over the way in which they were reached. Such a mechanism is provided by so-called ‘definitions’.

Following an idea of N.G. de Bruijn, we extend the notion of a context and allow contexts to contain definitions. Legal contexts that also contain definitions will be called ‘books’\footnote{De Bruijn cf. [de Bruijn, 1980] uses the term ‘book’ in a more general sense to refer to contexts that cannot only contain definitions but also local declarations. We restrict ourselves to the use of definitions only. For a discussion of the properties of books in the general sense, see: [Severi & Poll, 1994].}. The basic idea is the following: given the valid judgement $\Gamma \vdash E : T$, where ‘$E$’ is some compound expression, one can extend the valid context $\Gamma$ with a definition: ‘$x := E : T$’, where ‘$x$’ is a fresh variable. This definition expresses two things. First, that $\Gamma \vdash E : T$. Second, that one may, later, refer to the expression ‘$E$’ (which has the type $T$), using the symbol ‘$x$’. In essence, definitions function as abbreviations, and can be used later to form other, more complex objects. A ‘book’ will now look as follows:

\[
x_1 : T_1, \\
x_2 : T_2, \\
x_3 := E_3 : T_3, \\
x_4 : T_4 \\
. \\
. \\
. \\
x_n := E_n : T_n
\]
2.4. **BOOKS AND DEFINITIONS**

and represents a structured collection of assumptions, intermingled with conclusions that have been drawn based on them. These conclusions are still supported by their original justifications. So when we have a definition $x_3 := E_3 : T_3$, we can find within the expression $E_3$ all the assumptions that have been used to construct the object of type $T_3$. Note that the place of a definition in a book is important. A given definition can only depend on assumptions that precede it. In the sequel, the meta-variables $\Gamma$ and $\Delta$ will also range over books.

### 2.4.2 Some useful definitions

We will now introduce a number of useful definitions. These will also illustrate how one tends to work within a system like $\lambda C$. For ease of reading, we will adopt the convention that the binders ‘$\Pi$’ and ‘$\lambda$’ will be allowed to bind several variables of the same type. E.g. we write: ‘$(\lambda x, y.t.M)$’ instead of: ‘$(\lambda x.t.\lambda y.t.M)$’. We will also use conventional symbols as abbreviations for the definitions that we make, and use normal typographical conventions, writing binary connectives as infix-operators, etc.

**Absurdity**

Absurdity or ‘false’ is the type corresponding to the proposition which does not admit a proof. We will use the symbol ‘$\bot$’ to refer to it. If one has a proof of absurdity, one can prove every proposition. This is the famous principle: ‘Ex falso quodlibet’. Absurdity can be defined in $\lambda C$. The definition of absurdity is:

$$\bot := (\Pi p:* . p) : *$$

This definition does indeed have the property, that, if we have an inhabitant of the proposition $\bot$, then we can use it to construct an inhabitant of any proposition $A$. For, as the reader may verify, given the following book $\Delta$:

$$A : *, \bot := (\Pi x:* . x) : *, \text{paradox} : \bot$$

One has the judgement:

$$\Delta \vdash (\text{paradox } A) : A$$

Of course, as long as a context is consistent, it is not possible to construct an inhabitant of $\bot$. 
Logical connectives

Not all definitions in a book have to be at the level of types or objects. It is also possible to make definitions at the level of constructors. This is useful, for instance, to define the various logical connectives. In λC this can be done as follows:

- Implication, denoted by the symbol ‘⇒’, is defined in terms of ‘Π’ i.e. the dependent product, by:

\[ \Rightarrow := (\lambda p, q\!\!\!. (\Pi x:p.q)) : (\Pi p, q\!\!\!.) \]

As the reader can check, expanding the definition, the expression \((A \Rightarrow B)\) reduces to \((\Pi x:A.B)\). Any inhabitant of this type is a function that yields a proof of \(B\), given a proof of \(A\).

- Conjunction, denoted by the symbol: ‘∧’, can be given a second order definition, quantifying over propositions:

\[ \wedge := (\lambda p, q\!\!\!. (\Pi x:.(p \Rightarrow q \Rightarrow z) \Rightarrow z)) : (\Pi p, q\!\!\!.) \]

To see how this reproduces the intended meaning, note that, given a context where \(A : *, B : *\), the expression \((A \wedge B)\) reduces to: \((\Pi x:.(A \Rightarrow B \Rightarrow z) \Rightarrow z)\). This type will have the introduction and elimination rules for the conjunction. Given an inhabitant ‘\(p\)’ where \(p : (A \wedge B)\), both \(A\) and \(B\) can be proved using \(p\). Indeed, we have \(p : (\Pi x:.(A \Rightarrow B \Rightarrow z) \Rightarrow z)\), \((\lambda x:A.(\lambda y:B.x)) : (A \Rightarrow B \Rightarrow A)\) and \((\lambda x:A.(\lambda y:B.y)) : (A \Rightarrow B \Rightarrow B)\). Therefore, a proof of \(A\) is: \((pA(\lambda x:A.(\lambda y:B.x)))\). A proof of \(B\) is: \((pB(\lambda x:A.(\lambda y:B.y)))\).

Conversely, to prove \((A \wedge B)\) in a context where one has proofs \(a : A\) and \(b : B\), one can construct the object: \((\lambda z:(\lambda x:(A \Rightarrow B \Rightarrow z)(xab)))\) which is an inhabitant of \((\Pi x:.(A \Rightarrow B \Rightarrow z) \Rightarrow z)\) i.e. \((A \wedge B)\).

- Disjunction, denoted by the symbol: ‘∨’, can be defined in a way quite similar to conjunction:

\[ \vee := (\lambda p, q\!\!\!. (\Pi z:.(p \Rightarrow z) \Rightarrow (q \Rightarrow z) \Rightarrow z)) : (\Pi p, q\!\!\!.) \]

- Equivalence, i.e. bi-implication, denoted by the symbol: ‘⇔’, has an obvious definition in terms of implication:

\[ \Leftrightarrow := (\lambda p, q\!\!\!. (p \Rightarrow q) \land (q \Rightarrow p)) : (\Pi p, q\!\!\!.) \]

- Negation, given by the symbol: ‘¬’, is defined as implying absurdity:

\[ \neg := (\lambda p:* p \Rightarrow \bot) : (\Pi p:*\) \]
2.4. BOOKS AND DEFINITIONS

Equality

An example of a binary relation is equality. The definition of Leibniz- equality is that two things are equal if they are not distinguishable, i.e. if they share all their properties. This can be captured in the following definition, that quantifies over predicates, using the predicate-variable \( P \):

\[
\text{eq} := (\lambda x . (\lambda y . (\Pi P . (\Pi z . t . (P x) \Rightarrow (P y)))) : (\Pi t . (\Pi x . y . t . *)) )
\]

It is easy to prove that the relation defined in this way is indeed reflexive, transitive and symmetric.

2.4.3 Writing books

As mentioned before, mathematical textbooks can be formalised and written as type-theoretical ‘books’. The resulting books can be checked mechanically. To translate a particular mathematical theory, its basic entities are first introduced in the book, followed by the basic functions and predicates that play a role in the theory. The axioms of the theory are translated to propositions in the book, and fresh variables are introduced as inhabitants of these propositions. As they have inhabitants, the given propositions are now assumed true throughout the book. Logical connectives and the like can be introduced through definitions, as shown.

To prove a certain theorem within the book \( \Gamma \), the theorem has to be formulated as a proposition represented by a term \( T \) in this book, where \( \Gamma \vdash T : * \). If a term \( E \) can be constructed such that \( \Gamma \vdash E : T \), then the proposition \( T \) has been shown to be a theorem. To ‘remember’ the term \( E \) the book is typically extended with a definition, like ‘\( p := E : T \)’. In this way, the proof object \( E \) can be referred to later when more elaborate proofs are to be constructed that use \( T \) as a lemma.

Armed with the above definitions, type systems do indeed behave like logics. Without any further axioms, the resulting logics will be constructive. This means that the proofs that one constructs are computational: if the existence of a certain entity is proved, the proof provides a way to compute this entity. See, for instance [Greenleaf, 1991].

As is well known, not every formula that can be proved classically can also be proved constructively. For instance the formula \( (\neg \neg A \Rightarrow A) \) cannot be proved. If one wants the type system to mimic a classical logic, one has to add the double negation law. This means one has to add an axiom which indicates that for every proposition a proof of the double negation of this proposition leads to a proof of this proposition itself: \( \text{dneg} : (\Pi A : *. \neg \neg A \Rightarrow A) \). One cannot give a computational realisation for double negation, as, in general, the proof that a type cannot be empty does not provide us with an algorithm.
to compute an inhabitant of this type. For this reason the computational nature of those proofs that contain ‘\text{d}neg’ is destroyed.

The constructivist flavour of type systems is not so surprising, as these systems are based on $\lambda$-calculus, which is a calculus of \textit{computational} functions. From a computational viewpoint, a function is more than just a relation $R$ such that: $\forall x \exists! y. R(x, y)$; it is a way of computing a result given an element of the domain. Type systems reflect this computational view on functions, they treat functions as algorithms, rather than in an extensional way. For, as we remarked earlier, the only \textit{system} equality is $\beta$-equality. If two functions $f$ and $g$ are \textit{extensionally} equal, that is, if they both compute the same value for every argument in their domain, but are not $\beta$-equal then the type system will distinguish them: if $\Gamma \vdash E : T$ one most certainly does \textit{not} have $\Gamma \vdash (E : T)[f/g]$. So the type system distinguishes different ways to compute the same function.

This computational way of looking at things, that is characteristic for the constructivist mindset, is not limited to functions, but also applies to propositions. The interpretation of propositions is not extensional either, and propositions are not interpreted as truth values. The constructive interpretation – that is somewhat controversial in mainstream mathematics – is that the meaning of a proposition lies in the ability to recognise its \textit{proofs}. For instance, Heyting describes the meanings of the logical connectives by such explanations as: “A proof of $(A \land B)$ is a pair consisting of a proof of $A$ and a proof of $B$.” “A proof of $(A \Rightarrow B)$ is a function which maps each proof of $A$ into a proof of $B$” etc. See [Heyting, 1956].

One’s understanding of a proposition (the extent to which one knows what it means) thus depends on one’s ability to recognise a proof of that proposition. This view is known as the ‘Brouwer-Heyting-Kolmogorov interpretation’. In type-theoretical terms, it implies that the meaning of a proposition is given by the ability to recognise its inhabitants.

\section*{2.5 \textit{\Sigma}-types and inheritance}

\subsection*{2.5.1 \textit{\Sigma}-types}

A common extension of type systems allows us to form pairs and treat them as first-class citizens. The type of a pair is a so-called ‘dependent sum’ or a $\Sigma$-type. Roughly, dependent sums are generalisations of the conjunction, in the same way as dependent products are generalisations of the implication. To accommodate dependent sums, four extra rules are needed: a sum formation rule, a pair-introduction rule, and two pair-elimination rules, a ‘left’ one and a ‘right’ one. The formation rule for dependent sums, the ‘sum’ rule, is analogous to the product rule, but produces $\Sigma$-types instead of $\Pi$-types. The
2.5. Σ-TYPES AND INHERITANCE

rules for product formation were parametrised by a collection \( \mathcal{R} \); similarly, the rules for Σ-formation are parametrised by a collection: \( \mathcal{R}_\Sigma \). Given such a collection \( \mathcal{R}_\Sigma \), the rule for sum-formation is:

\[
\begin{align*}
\text{sum} & \quad \Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \\
& \quad \Gamma \vdash (\Sigma x : A.B) : s_3 \\
& \quad (\text{if } (s_1, s_2, s_3) \in \mathcal{R}_\Sigma)
\end{align*}
\]

Next we need a rule to create inhabitants of Σ-types, which are in fact pairs. The rule of pair formation is:

\[
\begin{align*}
\text{pair} & \quad \Gamma \vdash m : A \quad \Gamma, x : A \vdash B : s \\
& \quad \Gamma \vdash n : B[m/x] \\
& \quad \Gamma \vdash \langle m, n \rangle : (\Sigma x : A.B)
\end{align*}
\]

To eliminate pairs, we have the following two projection rules.

\[
\begin{align*}
\text{left} & \quad \Gamma \vdash M : (\Sigma x : A.B) \\
& \quad \Gamma \vdash \pi_1(M) : A
\end{align*}
\]

\[
\begin{align*}
\text{right} & \quad \Gamma \vdash M : (\Sigma x : A.B) \\
& \quad \Gamma \vdash \pi_2(M) : B[\pi_1(M)/x]
\end{align*}
\]

If pairing is followed by projection, the resulting construction can be simplified. This process is analogous to β-reduction where abstraction followed by application can be simplified. The reduction rules associated with pairing and projection are: \( \pi_1\langle M, N \rangle \rightarrow M \), and: \( \pi_2\langle M, N \rangle \rightarrow N \). Of course, the conversion rule should be adapted to include these new reductions, widening the equivalence relation \( \equiv_\beta \). With these rules, the resulting system is again strongly normalising\(^9\). For a proof, see [Geuvers, 1995].

2.5.2 Inheritance

Sometimes one wants to consider subtypes that are somehow formed by restricting a given type. This is useful because such restrictions of types occur in natural ontologies. For instance, animals are a special kind of creatures, mammals are a special kind of animals, horses are a special kind of mammals. Similarly, artifacts are special things, tools are special artifacts, and a saw is a special kind of tool. To express such relations, one would like to be able to ‘include’ a more specific type in a more general type. This should work in such a way that one can apply inhabitants of the more specific type in every case where inhabitants of the more general type might be applied.

\(^9\)Our choice for the collection \( \mathcal{R}_\Sigma \) is the following: \(<*,*,*>, <*,\square,\square>, <\square,*,\square>\) and \(<\square,\square,\square>\). With this choice the important system properties like strong normalisation are preserved. See for instance [Geuvers, 1995].

\(^{10}\)To assure this, pairs that are formed should retain information about their types. For a discussion, see [Luo, 1994, p. 31]
For example, everything that is valid for tools in general, definitely applies to a saw. To express formally that saws are just ‘special’ tools, we will allow declarations of the form:

\[ \text{saw} < \text{tool} : * \]

The effect of this declaration is that a type ‘saw’ is declared, and that any inhabitant of the type ‘saw’ is also an inhabitant of the type ‘tool’. We only allow such declarations for atomic types. Formally, in a context \( \Gamma \) which contains this declaration, and in which the judgement

\[ \Gamma \vdash s : \text{saw} \]

is valid, we also have the valid judgement:

\[ \Gamma \vdash s : \text{tool} \]

Of course, inheritance is transitive. If, in the context \( \Gamma \), a tool has been declared as a special ‘artifact’ by the declaration:

\[ \text{tool} < \text{artifact} : * \]

then the judgement:

\[ \Gamma \vdash s : \text{artifact} \]

must also be valid.

How can all this be achieved? Using \( \Sigma \)-types, the type ‘saw’ might be understood as an abbreviation for the type:

\[ (\Sigma t : \text{tool}. (\text{sawlike } t)) \]

Under this interpretation, the introduction of a variable \( s \) of type ‘saw’ amounts to the introduction of a pair consisting of a tool and a proof that the tool is ‘sawlike’. So for this pair one has: \( \pi_1(s) : \text{tool} \) and \( \pi_2(s) : (\text{sawlike } \pi_1(s)) \). Thus, the declaration

\[ \text{saw} < \text{tool} : * \]

can be seen as a definition:

\[ \text{saw} := (\Sigma t : \text{tool}. (\text{sawlike } t)) : * \]

If the type saw has an inhabitant \( s \), we can now, in cases where a tool is needed, use \( \pi_1(s) \), and in cases where a saw is needed, we can use ‘\( s \)’ itself. In other words, though we write the judgement: \( \Gamma \vdash s : \text{tool} \), this can be
2.6. THE MEANING OF TYPES

considered as a shorthand for: $\Gamma \vdash \pi_1(s) : \text{tool}$. If inheritance is used in this way, it is not a real extension of the system, but only a form of syntactic sugar.\footnote{This is important, because it assures that the system retains its pleasant properties.} Of course, inheritance between types is extremely useful in practical cases, as it is rather cumbersome to create ontologies where all the inheritance relations have to be coded explicitly.

For a precise and formal treatment of inheritance, see: [Barthe, 1996], [Betarte & Tasistro, 1998], [Luo, 1999]. For a formal application and implementation of inheritance (in a formalisation of Algebra), see: [Bailey, 1998].

2.6 The meaning of types

What does it mean that a term is an inhabitant of a certain type? What is the intuition behind types? Though we have seen that type theory can be used to mimic different logics, this does not answer the question about the nature of the types themselves. In this section we will address this question in an informal manner, in the hope to provide the reader with a semantic intuition for types.

As types have inhabitants, it is tempting to consider them as sets, assuming that the ‘inhabitant’ relation ‘$:\$’ is akin to the ‘element’ relation ‘$\in$’ in set theory. Though such a view can be fruitful, it may also be somewhat misleading, as there are profound differences between set theory and type theory. In particular:

- Set theory is static (there is a fixed universe) whereas type theory is dynamic (contexts are extensible).

- Set theory is extensional, whereas type theory is computational.

To understand the gap between the two ways of thinking, it may be instructive to try to interpret types as sets, and see why this is problematic. Let’s consider the idea that a type is just the set of all its inhabitants. There are several problems with this idea:

- It is problematic to determine the set of all terms that inhabit a given type. Whether a given type is inhabited by some term depends on the current context. But this context is never fixed, so this set is in general undefined.

- There is a difficulty with the identity of the ‘elements’ of this ‘set’. Which of the terms that inhabit a given type are supposed to refer to the same elements? For instance, take a type system in which natural numbers have been introduced. Consider the functions given by the
following terms: \((\lambda x : \text{nat}.(\lambda y : \text{nat}.(x + y))): \text{nat} \to (\text{nat} \to \text{nat})\) and \((\lambda x : \text{nat}.(\lambda y : \text{nat}.(y + x))): \text{nat} \to (\text{nat} \to \text{nat})\). Taking the set-theoretical viewpoint, one would identify these functions. But these functions will not be identified in the type system. So, there is a sense in which type systems are rather fine-grained, and a mapping to an extensional framework tends to be ‘forgetful’.

- In set theory, there is only one empty set. In type systems however, empty types abound, and they are not necessarily identical.

How can we interpret types intuitively, if we are to take the above problems seriously? A fruitful way to develop an intuition for types is to regard them as specifications. The view that types can be seen as specifications is part of the TT folklore, and has been around for some time. See, for instance [Martin-Löf, 1985]. It is largely motivated by the observation that constructive proofs have computational content, and therefore can be ‘executed’. The idea is that a simple type like ‘nat’ corresponds to a datatype. An inhabitant of this datatype corresponds to a concrete representation of a natural number that can be used to compute with. Inhabitants of a function type like \((A \Rightarrow B)\) do not correspond to arbitrary ‘mathematical’ functions but to algorithms that compute an instance of the datatype \(B\) from every instance of the datatype \(A\). As the expressive power of the type system is so strong, it is actually possible to use these types as full-fledged specifications.

### 2.6.1 Specifications

Consider an equation like:

\[x^2 = x\]  \hspace{1cm} (2.7)

In order to find the real roots of this equation, one can use it as a kind of specification, i.e. as way to determine whether a certain number is a root. A realisation of this specification is a composed object, consisting of an algorithm \(a\) to calculate a number (where \(a \to 1\) or \(a \to 0\) in this case) and a reason why the number that is calculated in this way satisfies the specification (a proof that \(a^2 = a\)). Thus the realisation is more than just the solution, more than an expression to compute this number; it also shows that this expression does indeed compute a number that ‘fits’ the specification.

Another equation, that has the same roots, like

\[x^4 = x\]

is thus a different specification. It does not have the same realisations, as the realisations in this case contain other proofs.
2.6. THE MEANING OF TYPES

Specifications are interesting concepts, because one can reason about their realisations in a purely hypothetical way, showing that a certain specification can be met, and even how it can be met, if realisations of certain other specifications are assumed.

Note that it is quite meaningful to consider the possibilities that follow from the availability of realisations for certain specifications, even if there is no guarantee that these realisations do exist in any way. The whole idea is not unlike that of a ‘Gedankenexperiment’, but it is somewhat more extreme. It is a ‘Gedankenexperiment’ in which the assumptions that are entertained do not even have to be logically possible. The conclusions that we draw are purely hypothetical.

To illustrate that specifications of hypothetical objects are meaningful, we consider another example. Suppose we are looking for the real roots of the equation:

\[(x^2 - 2 \times x + 2) = 0\]  

(2.8)

It is a quite clear and even fully operational specification of the thing that we are looking for; given a candidate solution, it is easy enough to see whether it satisfies the requirement. This specification is so clear, that it can be used to search for a solution in a systematic way. For instance, we may try to complete the square: substituting \(y = x - 1\), we find that any real root of the equation

\[y^2 + 1 = 0\]  

(2.9)

can be transformed into a root of the original equation. Therefore, when looking for a root of Equation 2.8, we can also search for roots of equation 2.9. Again, it is clear what is meant by a root of the equation \(y^2 + 1 = 0\). So the reasoning process that we go through is unquestionably meaningful. Nevertheless, there are no real roots for either equation. But the fact that a given specification cannot be met, does not make the specification meaningless, or invalidate any of the hypothetical results that the agent has derived. The fact remains, that had their been roots of 2.9, there would also have been a way to construct roots of 2.8. In fact, it is also known precisely how to effect such a construction. So it is possible to take these specifications seriously, and treat them as first-class citizens\(^\text{12}\). Doing so, we see that there is no easy way to capture specifications in extensional terms. In particular, there are countless specifications that have no realisations, but – from an extensional viewpoint – there is of course only one empty set. If types are interpreted as sets, therefore, all the specifications that have no realisations are jumbled together.

\(^\text{12}\)One way to introduce imaginary (sic) numbers, is by simply postulating them as solutions to such a specification.
2.6.2 Types as specifications

Specifications are ‘ways to recognise’ that which satisfies them, i.e. their realisations. Realisations are the things that fit the specifications. When an algorithm is seen as fitting a particular specification, it gives rise to an inhabitant of the corresponding type. This leads to the following interpretation of types and their inhabitants:

- Types correspond to ‘specifications’.

- An inhabitant of a type corresponds to a ‘realisation’ which includes not only an algorithm, but also some justification why this algorithm ‘satisfies’ or ‘fits’ the specification.

So, the inhabitant of a type correspond to a ‘realisation’ of this specification. As we have seen, a realisation is much more than just the naked algorithm: it will also contain an extra element (roughly, a proof) that ‘shows’ why an algorithm satisfies this particular specification.

The type theoretical context allows us to ‘declare’ that certain specifications have realisations, even though we are not able to construct these (yet). Of course, such assumed realisations cannot be ‘opened up’ to look what is inside them. They are like a black box\textsuperscript{13}. The black box nature of the declarations in a context gives type systems the power to talk about hypothetical objects. Essential about these objects is not what they are, but what is known about them, i.e. which specifications they meet. In fact, as types can be empty one can just as well talk about nonexistent and even impossible objects. In a consistent type system, there can be an arbitrary number of empty types. These types represent different specifications that have no realisations. The type system preserves the distinctions between all specifications, also between ‘unsatisfiable’ specifications. Within a consistent context, – i.e. a context that does not have an inhabitant for \( \bot \) – specifications of logically impossible constructions like ‘square circles’, or ‘planar maps that need more than four colours to be consistently coloured’ can never have inhabitants. So it is true that the corresponding types are all empty, but the system does not identify these empty types. It is simply not true that ‘planar maps that need more than four colours’ are ‘square circles’.

Recognising a planar map that needs more than four colours to be coloured is a rather different process from recognising a square circle. Within type systems, the identification of types only happens if they correspond to terms that are \( \beta \)-equal.

As a context can be extended with assumptions, it is possible to form a context that is logically inconsistent. But even in a context that is logically

\textsuperscript{13}In the other case where a context is extended with a definition, the realisation does exist and can of course be opened up.
2.7. Specifications and realisations

As we have explained, a type system enables us to decide whether a judgement of the form $\Gamma \vdash E : T$ is valid. In fact, it is even possible to compute a type $T$, if a legal term $E$ is given. This judgement can be interpreted as follows: given realisations of the specifications in the context $\Gamma$, the term $E$ is recognised to be a realisation of the specification as given by the term $T$.

This recognition problem is decidable, as we have seen in section 2.3.3.

Though it is useful and necessary to be able to recognise realisations of specifications, one often faces the following, more difficult, problem: one is given a context $\Gamma$, and a type $T$, such that $\Gamma \vdash T : s$, where $s$ is a sort. The problem is to construct a term $X$ such that: $\Gamma \vdash X : T$.

As a first example, consider the problem of constructing an inhabitant of type ‘nat’, in the context $\Gamma_6$, as given in section 2.2.3. This problem can readily be solved, and there are many such inhabitants. Some are in normal form: *one*, $(\text{sucone}), (\text{suc}(\text{sucone})) \ldots$ etc. Others are reducible, like: $((\lambda x: \text{nat} . (\text{suc}(\text{sucx}))) \text{one})$

Note, that a type like ‘nat’ is a *partial* specification that only specifies the datatype of its inhabitant. All that has been specified in this case is that the constructed entity is a natural number. To be more specific, one has to formulate more complex specifications. One way of doing this is through the use of $\Sigma$-types. For instance, consider the following type that is well-formed on the context $\Gamma_6$:

$$(\Sigma x : \text{nat} . (\text{odd}(\text{sucx})))$$

This type specifies a natural number whose successor is odd. An example of a realisation of this specification in the context $\Gamma_6$ is:

$$(\text{suc}(\text{sucone}), ((\lambda x_2. \text{one}) p))$$

It is possible to formulate more complex specifications, and very easy to create specifications that do not have realisations. For instance, consider:

$$(\Sigma x : \text{nat}. (\Sigma y : (\text{odd}(\text{sucx})). (\text{odd} x)))$$
This type specifies an odd number whose successor is also odd, and will not have a realisation.

2.7.1 Extending segments

Though a Σ-type can be used to represent a combined specification in a construction problem, it is often possible and far more convenient to replace this Σ-type by an equivalent, less complicated representation, a so-called extending segment:

**Definition 8 (extending segment)** A pseudo-context \( \Delta \equiv x_1 : T_1, \ldots, x_n : T_n \) is an extending segment of a legal context \( \Gamma \) iff \( \Gamma, \Delta \) is a legal context.

The combination of a given context with an extending segment of this context can also be used to represent a construction problem. The idea is that one has to find appropriately typed terms that can be substituted for all the variables that are declared in the segment. If one is able to construct terms within \( \Gamma \) that can be substituted for all the \( x_i \), a realisation of the extending segment has been found. This is made more precise in the following definition:

**Definition 9 (realisation)** Let \( \Delta \equiv x_1 : T_1, \ldots, x_n : T_n \) be an extending segment of \( \Gamma \), and let

\[ \Gamma \vdash D_1 : T_1 \]
\[ \Gamma \vdash D_2 : T_2[D_1/x_1], \text{ and} \]
\[ \Gamma \vdash D_3 : T_3[D_1/x_1, D_2/x_2], \text{ and} \]
\[ \text{... and} \]
\[ \Gamma \vdash D_n : T_n[D_1/x_1, D_2/x_2, \ldots, D_{n-1}/x_{n-1}] \]

then the substitution \([D_1/x_1, D_2/x_2, \ldots, D_{n-1}/x_{n-1}]\) is a satisfying substitution for \( \Delta \) in \( \Gamma \), and \( \Delta^* \equiv x_1 := D_1 : T_1, \ldots, x_n := D_n : T_n \) is a realisation of \( \Delta \) in \( \Gamma \) under the substitution \([D_1/x_1, D_2/x_2, \ldots, D_{n-1}/x_{n-1}]\).

It is easy to see that extending segments are related to Σ-types. A Σ-type \( T \) that is well-formed given some context \( \Gamma \), can be translated into a corresponding extending segment \( \Delta_T \). Under this translation, any realisation of \( T \) gives rise to a realisation of \( \Delta_T \) and vice versa. The type: \( (\Sigma x : A.B) \) corresponds to the extending segment

\[ y_1 : A, y_2 : B \]

A realisation of this segment, for example:

\[ y_1 := a : A, y_2 := b : B \]

leads to a realisation of the Σ-type, to wit: \( \langle a, b[a/y_1] \rangle \).
2.7. SPECIFICATIONS AND REALISATIONS

2.7.2 Top-down construction

Extending segments are also convenient for constructing realisations of specifications in a top-down fashion. Consider the problem of constructing an inhabitant for $T$ in a context $\Gamma$. In order to solve this problem, one may first try to construct a realisation for $T$ in an extended more ‘powerful’ context $\Gamma, \Delta$. This is rational, provided one has reasons to believe that the problem of realising the ‘intermediate’ specification $\Delta$ in the context $\Gamma$ is ‘easier’ than the original problem.

If the problem of constructing a $T$ can be solved on the extended context $\Gamma, \Delta$, the original problem has been ‘refined’ to the problem of the construction of a realisation for $\Delta$ on $\Gamma$. Using the substitution lemma the following fact can be proved:

**Fact 5** let $\Gamma, \Delta \vdash A : T$ and let $\Delta$ have a realisation in $\Gamma$, under the substitution $[S]$. Then $\Gamma \vdash (A : T)[S]$

This fact shows how a realisation for the ‘intermediate specification’ $\Delta$ in the context $\Gamma$ can eventually be used to construct a realisation of the original specification $T$ in the context $\Gamma$.

Formally, the idea that a context allows a problem to be refined in a certain way can be captured in the following definitions:

**Definition 10 (clause)** A clause is an expression of the form $[x_1 : T_1, \ldots, x_n : T_n] \Rightarrow A : T$ where $x_1 : T_1, \ldots, x_n : T_n$ is a pseudo-context, and $A : T$ a statement.

**Definition 11 (valid clause)** Let $x_1 : T_1, \ldots, x_n : T_n$ be an extending segment of $\Gamma$, and let $\Gamma, x_1 : T_1, \ldots, x_n : T_n \vdash A : T$. Then $[x_1 : T_1, \ldots, x_n : T_n] \Rightarrow A : T$ is a valid clause on $\Gamma$.

There are two reasons why clauses are useful:

The first reason is notational. The fact that a context has an inhabitant of a (nested) $\Pi$-type can be rewritten to a valid clause. The resulting clauses are far easier to read then the judgement that involves the $\Pi$-type. This kind of rewriting is sometimes referred to as the ‘heuristic application principle’.

The idea is to replace an inhabitant of a function type, by a clause that expresses that one has an inhabitant of the range of the function, under the assumption that one can find appropriate arguments, i.e. inhabitants for the domain type of the function. For example: Instead of $a : (\Pi x. (\Pi y. A))$ one writes $[t : \ast, y : t] \Rightarrow (a \, t \, y) : A$.

---

14 Barendregt, private communication.
The justification behind this rewriting is the following: if in a context \( \Gamma \) we have

\[ \Gamma \vdash a : (\Pi t : \ast. (\Pi g t : A)) \]

then it follows that:

\[ \Gamma, t : \ast, y : t \vdash (a \ t \ y) : A \]

In other words, the clause \([t : \ast, y : t] \Rightarrow (a \ t \ y) : A\) is valid in \( \Gamma \).

The second reason why clauses are useful, is that machines can use these clauses when searching to construct a realisation of a given specification. Using some (simple approximation of) higher order unification, and a special provision to deal with the ‘abstraction’ rule, one can construct backward-chaining algorithms that use clause resolution in an attempt to automatically construct realisations of a given specification. These algorithms are based on the fact that the resolution of valid clauses again yields (a partial instantiation of) a valid clause. Though the general problem of constructing realisations for a given specification is of course undecidable, it is not impossible to construct algorithms that perform reasonably well, at least in certain special cases. The interested reader is referred to [Helmink & Ahn, 1991] and [Helmink, 1992].
Chapter 3

Knowledge and observation

*A man's mind, once stretched by a new idea, never regains its original dimension.*
Oliver Wendell Holmes, Jr.

In this chapter we show how to use type-theoretical contexts; or more precisely, books, to represent the information of an agent about an external reality. We show how such books can be upgraded to meaningful information states which are related to an external reality by grounding them semantically. The kind of semantic grounding that we propose is not an abstract relation between a context and a mathematical model of the external reality, but it is a computational relation that is founded in the agent's *personal* ability to recognise certain types observationally. This idea opens up the possibility for knowledge and observation to truly interact, thus making the knowledge 'testable', or to be more precise, 'refutable'. We show that the resulting information states have many interesting and useful properties. Apart from being partial and computational — as they are based on inference — they are also extensible, as they are based on type theory. More importantly, due to the subjective nature of the semantic grounding, these knowledge states are intrinsically intensional. For instance, they can deal in a meaningful way with non-existent 'objects', and also allow an agent to distinguish different views on the 'same' object. Finally, we show that such information states lead naturally to a dynamic conception of knowledge in which reasoning and observation processes can be combined with a foundational approach to belief revision.

3.1 The problem

As we have seen, type theory (TT) is an attractive setting to capture mathematical reasoning. It allows us to detect what follows from given assumptions. We now want to employ TT to represent the knowledge that an agent has
about the reality that surrounds him. This means that we have to bridge
the gap between pure mathematical reasoning on the one hand, and applied
reasoning that deals with an external world, on the other hand. To do so,
we have to ascertain two things:

- The TT context, i.e. the body of assumptions that the agent is using,
must reflect the agent’s information about an external world.

- The types that play a role in such a context must have an interpretation
that causally connects them to this external reality. In other words, we
must assure that types that occur in the TT book are more than just
abstract syntactic symbols, and are somehow semantically grounded in
this external reality.

If we can assure that both conditions are met, the assumptions in a context
in TT can be used by the agent to reason about the reality surrounding him,
and are related to this reality. This means that the knowledge of the agent
can be applied in his dealings with the surrounding reality, and may also
be tested by his observations, creating the possibility for the agent to have
feedback on the quality of his knowledge about the world.

3.2 A book of knowledge

Our first question is how a book in type theory can be used as a formal
description of the information that an agent has about an external world.
The (standard) answer is that we use type theory as a ‘logical’ framework in
which the agent’s information about an external world can be stated. So the
information is represented in a context, and the agent subsequently reasons
in this context. This context, which will allow different models, acts as a
partial description of the external world.

An important complication is that the external world that surrounds an
agent is almost always dynamic. It is subject to changes. This raises the
question how one can formalise a changing world in a monotonic formalism.
The question will be investigated and answered in later chapters.

For the time being, we consider the situation where the external world
is fixed and immutable, like a giant storehouse of a company that is out of
business for eternity. Throughout this thesis we refer to this storehouse as
the ‘universe’. The problem of the agent is to make a useful inventory of this
universe. This inventory should later enable the agent to reason and answer
questions about this universe. Typical questions that one wants to answer
are about the existence of certain objects, or about regularities that can be
observed in this universe. To most of these questions there is no a priori
answer. The universe that we consider here is not a logical or set-theoretical
3.2. A BOOK OF KNOWLEDGE

universe, but a ‘real’ one. What can and what cannot be found in such a
universe is a contingent matter. To answer such questions, the agent needs
knowledge about this universe.
Within the knowledge of an agent one can distinguish several layers, re-
lecting the ontology of the universe under consideration. Of course, they
should not be confused with the different levels of the type hierarchy. It is
enlightening to distinguish at least four layers:

- First the layer of the logical groundwork, which enables the agent to
deal with logical connectives, quantifiers and equality. This logical
foundation can be defined in TT, as shown in the previous chapter.

- Next, certain mathematical notions have to be introduced. For in-
stance, it is important the agent knows that there are such things as
quantities, and that quantities can be compared and ordered. Also, the
agent must know that the resulting ordering relation is transitive, etc.
To express this in type theory, one has to create types that represent
the mathematical concepts, and introduce inhabitants for the relevant
axioms.

- The third layer constitutes an inventory of the kinds of objects that
the agent finds in the universe, and of the properties and relations that
the agent uses to classify and distinguish them. This layer provides the
conceptual framework in which different objects with their properties
and relations have to be made to fit. To create this framework formally
in TT the agent has to declare various types. These types can be
combined to create specifications of the various objects or state of affairs
that the agent can encounter in the universe.

- Finally, we need an inventory of the objects and situations that are
actually found in the universe. It is the layer of contingent facts. When
there is an object of a certain kind or a state of affairs that satisfies
a certain specification, the corresponding type — that can be formed
using the types that are present in the third layer — can be declared
to be inhabited. If certain rules or regularities are found to hold, the
types that express these are inhabited, and so on.

By accumulating these four kinds of knowledge one creates a knowledge
state. Note that, though we have sketched this context-building activity as
consisting of four stages, it is not the case that the more basic stages have to
be finalised before any of the work in the later stages can begin. One always
has the possibility to extend the conceptual framework, or even to extend
the logical or mathematical basis of the context, even though one has already
started to catalogue the different objects that are found in the universe. The rules of context formation will prevent any accidents from happening here.

It is on the basis of the resulting knowledge state that the agent is later able to answer questions about the universe. To clarify the whole process, it is best to take a small example.

### 3.3 Example

We consider an agent who reasons about different structures that are found in a universe. This universe is a part of cyberspace\(^1\), it contains at least two kinds of things, robots and tiles. (In subsequent chapters we will extend the example and allow the robots to move over these tiles and to paint them in different colours.) These things are characterised by different attributes. It is assumed that all things have a position and a colour. Robots also have a length, and a mood. Some of these attributes are quantitative, and can be compared. Other attributes can be decomposed in terms of simpler attributes. For instance, a colour might be decomposed in terms of ‘red’, ‘blue’ and ‘green’ intensities.

The information that the agent \( A \) has about the universe around him is represented in a Type Theoretical book, \( \Gamma_A \). What should this book look like? Though the book is a whole, we have already seen that it is convenient to consider it as being built in several stages that correspond to consecutive layers. The more basic layers also are the most general ones, and the ones that can be most often reused when different agents are designed. For instance, when one is constructing such agents, it makes sense to use ‘standard’ modules for logic and mathematics. Also, agents who live in similar worlds, i.e. in the same kind of environment in cyberspace, will be able to share large parts of their conceptual knowledge. In fact, if the agents live in the same world, they may even share factual knowledge. As humans, we are ourselves agents who share the same environment, and we tend to find those aspects of our knowledge that we share with all others less interesting. It is, literally, commonplace. However, if we try to understand the structure of the knowledge state of artificial agents in more detail, it is important to see how the more specific knowledge it rooted in the more basic knowledge. Therefore, we have no choice but to drag the reader through the somewhat boring exercise of setting up and examining all the layers that are involved.

\(^1\)For what follows in this chapter, as opposed to the following ones, that deal with time and change, it is by no means essential that the universe that the agent deals with is a part of cyberspace. However, when we assume this, it is somewhat easier to imagine how various observational abilities that we attribute to the agent can indeed be realised. This avoids undue speculations about the possibility or impossibility of certain observation mechanisms.
3.3. **EXAMPLE**

That is what we will do in this example.

### 3.3.1 The logical layer

The first and most basic layer is the part which contains the definitions of the 'logical' concepts. In this layer we find the definitions of logical concepts like: ‘⊥’, ‘⇒’, ‘¬’, ‘∨’, ‘eq’, etc. These have already been introduced in section 2.4.2. This part of the context might start as follows:

\[
\begin{align*}
\bot & := (\Pi p^*, p) : \ast, \\
\Rightarrow & := (\lambda p, q^*.(\Pi x.p.q)) : (\Pi p, q^* . \ast) \\
\land & := (\lambda p,q^*. (\Pi z^* . (p \Rightarrow q \Rightarrow z) \Rightarrow z)) : (\Pi p,q^* . \ast), \\
\neg & := (\lambda p^*. (p \Rightarrow \bot)) : (\Pi p^* . \ast) \\
eq & := (\lambda t^*. (\lambda x, y^*. (\Pi P. (\Pi z^* . (P x \Rightarrow (P y)))))) : (\Pi t^*. (\Pi x, y^* . \ast)) \\
d\text{neg} & := (\Pi p^*. \neg p \Rightarrow p)
\end{align*}
\]

Note the inclusion of the double negation axiom. The agent reasons in classical logic. This is no problem; the agent will have many other assumptions for which he does not have a computational realisation.

### 3.3.2 The mathematical layer

In the second layer we find declarations of the various types that constitute the ‘mathematical’ foundation of the agent. Among other things, this layer may contain types that represent abstract concepts, like comparable quantities. To introduce the concept of quantities, one may, for instance, start by introducing the following ideas. (We use clause notation, as introduced in section 2.7.2.)

Some types correspond to quantities:

\[ [t : \ast] \Rightarrow (\text{quant } t) : \ast \]

Quantities of the same type can be ordered by an ordering ‘gt’:

\[ [t : \ast, p : (\text{quant } t), x : t, y : t] \Rightarrow (\text{gt } t p x y) : \ast. \]

The resulting ordering is total:

\[ [t : \ast, p : (\text{quant } t), x : t, y : t] \Rightarrow (\text{tot } t p x y) : (\text{gt } t p x y) \lor (\text{gt } t p y x) \lor (eq t x y) \]

It is antisymmetric:
\[ t : *, p : (quant t), x : t, y : t, q : (gt t p x y), r : (gt t p y x) \]
\[ \Rightarrow (\text{anti } t p x y q r) : \bot \]

It is transitive:

\[ t : *, p : (quant t), x : t, y : t, z : t, q : (gt t p x y), r : (gt t p y z) \]
\[ \Rightarrow (tr t p x y z q r) : (gt t p x z), \]

\[ \ldots \]

The agent may also know natural numbers, these one might introduce as follows:

\[ nat : *, \]
\[ zero : nat, \]
\[ suc : nat \rightarrow nat, \]

Numbers are quantities

\[ qn : quant(nat), \]
\[ [n : nat] \Rightarrow (step n) : (gt nat qn (suc n) n), \]

natural numbers are closed under induction

\[ [P : nat \rightarrow *, q : (P \ zero), s : (\Pi x : nat. (\Pi y : P x). (P (suc x)))] \Rightarrow (\text{induc } P q \ s) : (\Pi x : nat. (P x)) \]

\[ 3.3.3 \text{ The conceptual layer} \]

In the third layer one finds types that represent more concrete concepts like ‘thing’, ‘robot’, ‘location’, ‘colour’ and the like. This is the layer of real-world concepts and predicates. For instance, one might introduce the following concepts.

There are things:

\[ thing : * \]
3.3. **EXAMPLE**

Robots and tiles are things:

\[\text{robot} < \text{thing} : \ast,\]
\[\text{tile} < \text{thing} : \ast\]

There are different kinds of attributes that may characterise these different things:

\[\text{colour} : \ast,\]
\[\text{position} : \ast,\]
\[\text{length} : \ast,\]
\[\text{mood} : \ast,\]
\[\text{intensity} : \ast\]

Of these attributes, at least lengths and intensities are quantitative

\[q : (\text{quant length}),\]
\[p : (\text{quant intensity})\]

Things have a position and a colour

\[\text{pos} : \text{thing} \Rightarrow \text{position},\]
\[\text{col} : \text{thing} \Rightarrow \text{colour}\]

A colour can be decomposed in different spectral intensities

\[r : \text{colour} \Rightarrow \text{intensity},\]
\[g : \text{colour} \Rightarrow \text{intensity},\]
\[b : \text{colour} \Rightarrow \text{intensity}\]

Robots have a length:

\[\text{len} : \text{robot} \Rightarrow \text{length}\]

There are also some predicates on the attributes. For instance, a given colour might be red, white or blue. The predicates over colour might be given definitions\(^2\) in terms of the different spectral intensities. For instance, we might define predicates ‘rr’, ‘gg’, ‘bb’ and ‘ww’ over colours that select colours that are red, green, blue or white:

\(^2\)To be realistic, these definitions would have to be somewhat more sophisticated.
\[ rr := (\lambda x : \text{colour.} \quad (\text{gt intensity} p (r x)(g x)) \land \\
(\text{gt intensity} p (r x)(b x))) \quad : (\text{colour} \rightarrow *) \]

\[ gg := (\lambda x : \text{colour.} \quad (\text{gt intensity} p (g x)(r x)) \land \\
(\text{gt intensity} p (g x)(b x))) \quad : (\text{colour} \rightarrow *) \]

\[ bb := (\lambda x : \text{colour.} \quad (\text{gt intensity} p (b x)(r x)) \land \\
(\text{gt intensity} p (b x)(g x))) \quad : (\text{colour} \rightarrow *) \]

\[ ww := (\lambda x : \text{colour.} \quad (\text{eq intensity} p (g x)(r x)) \land \\
(\text{eq intensity} p (r x)(b x))) \quad : (\text{colour} \rightarrow *) \]

Of course there are also predicates on objects: a robot may be longer than another robot, things may be judged to have different colours. Many of these predicates will have a definition in terms of quantitative attributes, or of predicates on attributes that reduce to conditions on quantitative attributes:

\[ \text{longer} := (\lambda x, y : \text{robot.} (\text{gt length} q (\text{len} x)(\text{len} y)) : (\Pi x, y : \text{robot.*}, \text{red} := (\lambda x : \text{thing.} (rr (\text{col} x))) : (\Pi x : \text{thing.*}), \text{grn} := (\lambda x : \text{thing.} (gg (\text{col} x))) : (\Pi x : \text{thing.*}), \text{blue} := (\lambda x : \text{thing.} (bb (\text{col} x))) : (\Pi x : \text{thing.*}), \text{white} := (\lambda x : \text{thing.} (ww (\text{col} x))) : (\Pi x : \text{thing.*}) , \ldots \]

3.3.4 The factual layer

In the highest ontological layer, we eventually find the inhabitants of the various types: individual objects, and proofs that certain objects have certain properties.

R2d2, Marvin and C3po are robots:

\[ r2d2 : \text{robot}, \text{c3po} : \text{robot}, \ldots \]
3.3.  **EXAMPLE**

*marvin : robot*

R2d2 is red:

*hearsay : (red r2d2)*

There is only one red robot:

\[
[r : robot, s : robot, q : (red r), p : (red s)] \Rightarrow (ax rs q p) : (eq robot rs)
\]

C3po is longer than R2d2:

*fact1 : (longer c3po r2d2)*

Marvin is longer than C3po:

*fact2 : (longer marvin c3po)*

These chunks of knowledge in the four layers together form a knowledge state, which really is a partial specification of the universe, according to the agent A. We call the resulting context \( \Gamma_A \). The reader should note that there can be many different knowledge states which encode knowledge about the same universe, because there can be many different specifications of one given thing. Different specifications emphasise different aspects, and it is to be expected that different agents might conceptualise the same universe in different ways.

### 3.3.5  **Drawing conclusions**

What can an agent A with an information state represented by a book \( \Gamma_A \) be said to know? This will ultimately depend on the extending segments for which he is able to construct realisations within his knowledge state.

An extending segment for a knowledge state \( \Gamma_A \) represents information that makes sense, given the knowledge in \( \Gamma_A \). If the agent can indeed construct a realisation for the extending segment, he is considered to believe this segment.

As we have explained, the crucial ability for the agent is the ability to construct realisations for extending segments of his knowledge state. It is illustrative to investigate some of the segments for which an agent armed with the above information state given by the book \( \Gamma_A \) can construct a realisation, and to see what kind of knowledge they imply.
Among other things, the agent can show that Marvin is longer then R2d2, by exhibiting an inhabitant for the following extending segment

\[ z : (\text{longer marvin} r2d2) \]

The associated proof object for ‘z’ is:

\( (\text{tr length q (len marvin)} (\text{len c3p0})(\text{len r2d2}) f\text{act2 fact1}) \)

One can also show that no robot is longer then itself, by finding an inhabitant for the following segment:

\[ g : (\Pi x : \text{robot} \neg(\text{longer x x})) \]

The associated proof object for ‘g’ is:

\( (\lambda y : \text{robot}.(\lambda f : (\text{longer y y}).(\text{anti length q (len y)}(\text{len y}) f f))) \)

Further one can show that Marvin is not R2d2, by finding an inhabitant for

\[ h : \neg(\text{eq robot marvin r2d2}) \]

The associated proof object for h is:

\( (\lambda x : (\text{eq robot marvin r2d2}),(g r2d2(x(\lambda y : \text{robot}.(\text{longer y r2d2})z)))) \)

With this, one can prove that Marvin is not red

\[ q : \neg(\text{red marvin}) \]

with the following proof object for ‘q’:

\( (\lambda s : (\text{red marvin}).(h (ax marvin r2d2 s hearsay))) \)

One can also show many other things, for instance, that red things are not green:

\[ x : (\Pi r : (\Sigma t : \text{thing} .(\text{red} t)).\neg(\text{grn} r1(r))) \]

This last segment can also be written in clause form as:

\[ t : \text{thing}, q : (\text{red} t) \Rightarrow (x t q) : \neg(\text{grnt}) \]
3.4. SEMANTIC GROUNDING

The construction of a realisation of this segment, i.e. of a proof object for ‘\(x\)’, is left as an exercise to the reader.

3.3.6 Summary

To sum up, what we have shown in this section is:

- One can provide an agent with a book that expresses his information about an external world.
- In this book one can combine logical, mathematical, conceptual and contingent information. The knowledge state is inherently partial.
- Given his knowledge state, the agent can deduce the consequences of this knowledge state.

All this seems very nice and obvious. However, we have cheated in a very important way. By themselves, the segments for which we constructed these realisations mean nothing. It still has to be explained how one is to relate the various terms in this knowledge state to an external world. This is perhaps not immediately apparent, as the meaningful names that were given to the variables in the knowledge state induce the reader to interpret these variables in a certain way. But the interest that these knowledge states may have gained in this way is strictly dependent on the interpretations that the reader almost automatically ‘projects’ on the different variables. Our next task will be to preserve this interest and to give an account of semantic grounding that explains how the agent himself is able to have similar interpretations of these variables.

3.4 Semantic grounding

In this section we show how type-theoretical contexts can be semantically grounded, in order to form information states that allow an agent to ‘understand’ an external world.

3.4.1 The problem

We face the following situation. On the one hand we have an agent, who has a knowledge state, represented as a context, or more precisely, a book, in type theory. On the other hand we have some, possibly virtual, reality. The question that needs to be answered is how these two sides can be related. More precisely, we want to know how the agent himself can relate his information to the external reality in a computational way.
CHAPTER 3. KNOWLEDGE AND OBSERVATION

What can we learn from the literature about this problem? Epistemic (modal) logics enable us to reason about knowledge and its relation to reality. These frameworks are quite powerful, and one is able to characterise certain properties of an agent’s information within it. For instance, veridical knowledge, i.e. knowledge that can be said to be true, is characterised by the T axiom:

\[ K(\phi) \Rightarrow \phi \]

The meaning of this axiom can be explained formally in terms of the reflexivity of Kripke frames. The axiom holds if the actual world is one of the reachable epistemic alternatives of this world. From a theoretical viewpoint this is indeed a perfect characterisation of veridical knowledge. Unfortunately, however, this axiom does not help an agent who wants to relate his own knowledge to an external reality. Modal logics enable us to understand and describe the relation between a world and the knowledge about this world, but they describe the whole situation (containing both the agent with his knowledge and the external reality) from an external viewpoint. This leads to an idealistic description, in which the knowing agent and the surrounding reality are seen from the viewpoint of an omniscient external observer. Though the resulting construction may indeed be said to relate the knowledge of the agent to the state of affairs in some world, this relation exists only in the eyes of an external logician-observer, and it is hardly a computational relation, and certainly not the kind of relation that can be computed by the agent himself. Epistemic logics are indeed discussing knowledge and reality, but they do some from an external viewpoint. What is needed instead, is an intrinsic form of epistemics, as advocated in [Benthem, 1991].

Another approach is to use model theory directly. Surely, a truth definition provides a link between theories in some logic \( L \) and a model \( M \) that resides outside this logic. Typically, in this case, there is some interpretation function ‘\( \phi \)’ that relates the terms of \( L \) that occur in the theory to a Herbrand model. This function has a precise recursive description, and provides a computational link between the logic and an external model. And even though we need a relation between a theory and some external world, and not between a theory and a Herbrand model, the basic idea, of using a recursive interpretation function, seems workable. The only problem is to adapt this solution in such a way that the restricted computational viewpoint of the agent is taken into account. For an agent can only use an interpretation function to link the terms of his logic to an external reality, if he himself has access to this interpretation function.

If he tries to use his context as a repository of information about the world, he must be able to apply the function ‘\( \phi \)’ that relates the ‘things’ in the real world to the variables that are declared in his context. The function
3.4. SEMANTIC GROUNDING

'\( \phi \)' must somehow have been 'built in' within the agent. But if this idea is taken seriously, one needs a programmer that, for each entity in the reality, sets up an explicit link to the corresponding variables in the agent's theory, c.q. the variables in his context. The programmer needs to know the whole world that the agent can encounter beforehand, and this is a very strong, and in many cases unrealistic constraint. Also, the resulting solution is inherently inflexible, and it is hard to see how — under such circumstances — the implicit information of the agent might ever be allowed to grow. If this solution is adopted, the agent can only play inside a small 'playground' that has been prepared for him, but once he gets outside of this 'playground' he loses his bearings. To conclude, this approach may have some merit, but it should somehow be supplemented with a mechanism that allows the agent to gather information all by himself.

3.4.2 Gathering information

Maybe we should try a completely different approach, and investigate the problem anew, starting with a very simple problem. Let us assume an agent who knows that there is a robot 'R2d2' and who also knows that this is the only red robot. The agent is looking for R2d2.

To come up with a computational solution to this problem, it is instructive to investigate it practically. Thus we first consider the question how the agent might identify R2d2 in practice. A moment's reflection shows that, in order to do so, it is rather important that the agent knows what a robot looks like, i.e. what kind of observable pattern a robot presents to an observer. If he does know this, this may help the agent to recognise robots in general. So, at a certain point in time he sees some object, this object looks like a robot, i.e. it presents the appropriate observational pattern, and so the agent takes this to be a robot. If the agent also has a way to see that this robot is red — for instance by evaluating the intensities of the light refracted from the surface of the robot at different wavelengths — it may indeed be possible for the agent to identify, not just a robot, but a red robot. Knowing that R2d2 is the only red robot, the agent is able to conclude his search successfully. All this is straightforward, and it is rather obvious how it might work in practice.

But if we accept this, and investigate what the agent has been doing, we see that he certainly did not apply some function to compute some entity in the real world from some constant 'R2d2'. What he has been using is a procedure that can recognise an external pattern using a description of R2d2. This procedure is not at all guaranteed to return with some value, and it is also not guaranteed to always return with the same\(^3\) value. This shows that

\(^3\)For how can the agent be sure that the robot that he has recognised is R2d2? This
an agent is not able to recognise individuals directly and has to rely on his ability to recognise 'kinds' or 'types'. So, from a computational viewpoint, an agent can realise a correspondence between his information and the real world, because he can recognise certain patterns within this world. What he recognises, therefore, are, first and foremost, kinds of things. I.e. things that satisfy some observational criterion. An agent’s recognition of individuals rests on certain ‘uniqueness’ assumptions.

This conclusion is not only in accord with common sense, and with basic ideas about cognition in psychology (See, for instance: [Estes, 1994]) but is also useful within a more theoretical debate. If it is accepted that knowledge is related to an external world because certain concepts that figure in this knowledge correspond to observational criteria, it is no longer necessary to refer to entities to give a valid account of meaning. This helps us to exorcise a few demons that haunt referential theories of meaning. For it seems that correspondence theories of meaning which explain meaning by referring to individuals in the external world, have to face the following problems:

- They cannot deal easily with nonexistent objects, and need elaborate constructions involving possible worlds in order to discuss them meaningfully.

- They cannot deal in satisfactory ways with the meaning of categories that are logically impossible.

- They cannot deal with observable phenomena that do not correspond clearly to external objects, — phenomena that can be meaningfully referred to, but which do not correspond with objective entities in any conventional sense, but involve a personal perspective on things — like ‘right’ and ‘left’, ‘come’ and ‘go’, ‘pain’ and ‘hunger’ or even ‘horizon’ and ‘rainbow’.

These problems seem to disappear in the approach that we advocate. To show this we investigate the view that meanings should eventually be grounded in terms of patterns that an agent can recognise. To do so, we first examine how such a view can be made operational, relating a type-theoretical context to an external reality, based on an agent’s observational abilities.

---

^is not possible. The agent’s recognition of this individual was founded on an assumption. This assumption is that there is no other individual that presents us with the same observational pattern. Indeed, if this assumption is not warranted, as when there are several red robots, a corresponding error of judgement may well occur.
3.5 Observation

How can the idea that one can relate TT books to an external reality, using an agent's ability to recognise patterns, be made sufficiently precise?

In a nutshell, the idea is as follows. There is an agent $A$, who has a current information state that is represented by a type-theoretical book $\Gamma_A$. The well-formed types that can be formed in this book correspond to the concepts that the agent uses to categorise and understand the external world. The agent has the ability to recognise inhabitants for some of these types. This means the agent can encounter observational patterns in the external world that he can interpret as exemplifying these types. So it is the agent himself – using his observational abilities – who establishes the connection between his information and the external world.

Note that, as a consequence of this view, the agent also has the ability to extend his information state on the basis of observations. Using the information in his context, he can reach new conclusions on the basis of the new facts. These new conclusions can also be stored in the knowledge state, and may or may not be observable themselves. In the long run, as the knowledge state is extended further, this process may even give rise to a situation in which a paradox is deductible. If this is the case, some of the agent’s knowledge is apparently unreliable, and he has to resort to a form of belief revision to cope with the situation.

3.5.1 Some definitions

To make all this more precise, a number of definitions are needed.

**Definition 1 (types)** The set $\Theta$ of all types that the agent uses is the set of all types that are well-formed in his context $\Gamma$, i.e.: $\{t | \Gamma \vdash t : \ast\}$

Note that ‘$\bot$’ is always an element of ‘$\Theta$’.

Among all the types the agent uses there are those types whose inhabitants might be observed in the external world:

**Definition 2 (observable types)** The agent has a set $\Theta_0$ of observable types. The observable types are a subset of all types: $\Theta_0 \subset \Theta$.

In what sense are the types that are elements of $\Theta_0$ observable? For a type $t \in \Theta_0$ it is conceivable that the agent ‘sees’ inhabitants. In other words, the agent has a method $M_t$ of recognising the type $t$. Whether this method, when applied, will ever lead to the actual recognition of an inhabitant is a different matter. Roughly, this depends on two things:

- Whether there is anything in the external reality that ‘corresponds’ to the type, i.e. whether the type is ‘potentially’ inhabited.
CHAPTER 3. KNOWLEDGE AND OBSERVATION

- Whether certain (non-specified) observational conditions are fulfilled. For instance, the agent has to be looking at the right place at the right time, etc.

So the method to observe a given type is nothing but a ‘test’ on reality that can either fail or succeed. If the test succeeds, an inhabitant of the type has been observed. If not, nothing follows.

The fact that the agent can relate observations of the universe to his theory of the universe also opens up the possibility to test such a theory. In certain cases the agent may be confronted with the fact that his knowledge is not accurate. This is captured in the following definition.

Definition 3 (refutation) A sequence of successful observations of the types $T_1, T_2, \ldots, T_n$ will refute a type theoretical book $\Gamma$ if there exists a term $E$ such that, for arbitrary fresh variables $x_1, x_2, \ldots, x_n$, one has: $\Gamma, x_1 : T_1, x_2 : T_2, \ldots, x_n : T_n \vdash E : \bot$.

The question how an agent can proceed in such cases is briefly considered in section 3.7.3.

3.5.2 Observable types

Which types are observable? In principle we are free to select those types that we like, and proclaim them to be observable ones, if only we are willing and able to provide a method to the agent to indeed recognise inhabitants of these types in the outside world. (This constraint is a result of the fact that we limit our discussion to computational systems). The question: ‘Which types are observable?’ can therefore be replaced by the question: ‘For which types does the agent have a method to recognise its inhabitants?’ The answer to this second question clearly depends on the construction of the agent. The reader can imagine that an agent might, when filtering its sensory input through neural networks or the like, recognise the inhabitants of types that correspond to nouns like ‘robot’, ‘unicorn’, ‘chair’, or ‘house’. So these types might be observable. But what about the inhabitants of propositions like: ‘Marvin is longer than R2d2’? Or: ‘The robot R2d2 is red’? Type theory tells us that the inhabitants of propositions are proofs of these propositions. Can an agent recognise those? The idea of recognition of proofs sounds familiar; it is the basic idea of the Brouwer-Heyting-Kolmogorov interpretation of proofs, that we encountered in chapter 2. This interpretation should somehow be stretched so as to include observational data. What might count as an — observationally founded — proof that: ‘Marvin is longer than R2d2’? Clearly, a measurement of the length of both Marvin and R2d2, followed by a comparison. What can count as an — observationally founded — proof that R2d2 is red? One possibility is a comparison of different spectral components.
of light reflected from R2d2's surface, which shows that observed intensities of certain lower frequencies are higher than the observed intensities of other, higher frequencies. So we can assume that an agent can recognise inhabitants of these propositions if he can make the appropriate observations.

Such considerations show that, depending on the way in which the agent is constructed, many types that occur in his knowledge state can indeed be observable. Of course, this should never be taken to imply that all these types can be observable. This is definitely not true. Typically, for instance, it is not conceivable how anyone or anything could (in general) observe the inhabitants of 'II' types, that correspond to universal quantifiers\(^4\).

As we discussed, whether certain types can be assumed to be observable by the agent \(A\) depends primarily on his construction. We can assume that an agent with specific observational abilities exists, if we know how we might computationally realise these abilities. There are countless possibilities here. For the sake of definiteness, we will, in the sequel, only consider agents who have the kind of observational abilities listed below\(^5\):

1. They can recognise certain kinds of objects like robots, houses, chairs, unicorns, machines or graphs.

2. They can compute an abstraction from the pattern that has been recognised. For example, given a house, they can discern its roof, given a unicorn, they can discern its horn, given a horn, they can discern its length, colour, orientation, etc. Given a graph, they can compute the number of its nodes, adjacency relations, etc.

3. As they can discern quantities like the 'length' of the horn of a unicorn, or the surface area of the roof of a house, they can compare comparable quantities. So they may be able to compare the length of a horn with a yardstick, or the surface area of a roof with the area of another surface.

4. They can combine the above observational abilities to a more complex procedure that allows them to recognise the inhabitants of \(\Sigma\)-types. In other words, they may have the ability to recognise robots that are red, unicorns that have a long horn, or a planar graph that needs 5 colours to be coloured consistently.

\(^4\)To some extent the situation that we sketch here, in which a theory is related to possible observations, is obviously subject to the kinds of constraints that we know from the Popperian analysis of the scientific process.

\(^5\)This list is motivated by the fact that it is clear how one can build agents who have the mentioned abilities. The list is not complete. For instance, it does not mention the fact that certain agents might have the ability to estimate probabilities using random tests.
3.5.3 Example

To see where such an analysis of observational abilities leads, we revisit the agent $A$ with knowledge state $\Gamma_A$ that we introduced in section 3.3. Concretely, applying the above criteria to the example at hand, we can give a recursive characterisation of the set of observable types $\Theta$, for the given agent $A$. From this characterisation we can subsequently derive which types are observable to $A$.

In order to do so, we first need to characterise all variables and terms that have an interpretation for $A$, due to the agent’s observational abilities discussed above. Inspecting the various terms, we see that:

- Given the observational abilities of the first kind, terms like `thing`, `robot` and `tile` have an interpretation.

- Due to the observational abilities of the second kind, terms like `(loc x)` and `(col x)` have an interpretation, provided that $x$ has an interpretation, and that the $x$ has type ‘thing’. Similarly, terms like $(r x), (g x)$ and $(b x)$ have an interpretation, if $x$ has an interpretation, and the type of $x$ is ‘colour’.

- Given the observational abilities of the third kind, that enable the agent to compare quantities, we feel justified in saying that terms like `(gt intensity p x y)` and `(eq intensity x y)` have an interpretation, provided the terms $x$ and $y$ have an interpretation, and are of type ‘intensity’. Also, the terms `(gt length q x y)` and `(eq length x y)` have an interpretation, provided $x$ and $y$ have interpretations, and are of type ‘length’. Finally, the terms `(gt nat q n x y)` and `(eq nat x y)` have an interpretation, provided $x$ and $y$ have interpretations, and are of type ‘nat’.

We want to define the set of observable terms. To get a formal grip on these terms, we need a formal definition of the set of interpretable terms. Generalising, we see that the terms that have an interpretation for an agent are formed by application from a number of interpretable primitives. For the agent $A$, this set is, in accord with the listing above:

$$\exists = \{\text{thing}, \text{robot}, \text{tile}, (\text{gt intensity } p \ x \ y), (\text{eq intensity } x \ y), (\text{gt length } q \ x \ y), (\text{eq length } x \ y), \text{loc}, \text{col}, r, g, b\}$$

Note that all the interpretable primitives are well-typed in $\Gamma_A$, i.e. for any $s \in \exists$, there exists a term $T$, with $\Gamma_A \vdash s : T$.

Given a context $\Gamma$ and a set of interpretable primitives $\exists$, we can define the set of interpretable terms.
3.5. OBSERVATION

Definition 4 (interpretable terms) Let \( \Gamma \) be a context and \( \mathcal{S} \) be a set of interpretable primitives that are well-typed in \( \Gamma \). A term \( t \) is interpretable on the context \( \Gamma \) given the set \( \mathcal{S} \), iff it is well-typed in \( \Gamma \), i.e. there exists a term \( T \), with \( \Gamma \vdash t : T \), and either \( t \in \mathcal{S} \) or \( t \) reduces to an application expression \( (t_a t_b) \) where both \( t_a \) and \( t_b \) are interpretable on the context \( \Gamma \) given the set \( \mathcal{S} \).

Given the definition of interpretable terms on a set \( \mathcal{S} \), we are ready to define all types that are observable given the set \( \mathcal{S} \).

Definition 5 (observable type) Let \( \Gamma \) be a context and \( \mathcal{S} \) be a set of interpretable primitives on \( \Gamma \). If \( T \) is an interpretable term on \( \mathcal{S} \), and \( \Gamma_A \vdash T : * \), then \( T \) is an observable type on \( \Gamma \), given \( \mathcal{S} \). Also, if \( T \) is observable on \( \Gamma \) given \( \mathcal{S} \) then the type \( (\Sigma x : T, P) \) is observable on \( \Gamma \) given \( \mathcal{S} \) iff \( P \) is observable on \( \Gamma, x : T \), given \( \mathcal{S} \cup \{x\} \). Moreover, if \( T_1 \) and \( T_2 \) are observable on \( \Gamma \) given \( \mathcal{S} \) then \( (T_1 \wedge T_2) \) is observable on \( \Gamma \) given \( \mathcal{S} \).

The idea is that the agent has, for each observable type \( T \), a method to recognise its inhabitants. The justification for the above definition is that is indeed possible to implement such a method for the types that are thus defined as observable.

3.5.4 Recognising individuals

If we take the idea seriously that all that an agent can recognise in the external world are ‘types’, one may wonder how the ability to recognise individuals — that many agents clearly exhibit — can ever arise from the ability to recognise types. The answer is that such individuals can be recognised because they are related to types that are assumed to have unique inhabitants.

An agent who has different inhabitants of a given type, and wants to treat these as separate individuals, has to assume that there is some well-defined aspect in which the different inhabitants of the given type are unique\(^6\). This assumption subsequently allows him to use this aspect to identify them. For example, if there are several inhabitants of type robot, the agent can distinguish them if every robot is furnished with a unique serial number.

Formally, the assumption that every robot has a serial number is expressed as follows:

\[
[r : \text{robot}] \Rightarrow (n r : \text{nat})
\]

\(^6\)Often, the agent faces a real and even very difficult problem, if he is not able to find such an identifying aspect. The problem can actually be extremely hard to solve, and is even more keenly felt in cyberspace, where objects, due to their digital nature, and the ease with which they can be copied, are sometimes difficult or even impossible to distinguish.
These serial numbers are unique:

\[ r : \text{robot}, \, q : \text{robot}, \, p : (\text{eq} \, \text{nat}(n \, r) \, (n \, q)) \]

\[ \Rightarrow (\text{uniq} \, r \, q \, p) : (\text{eq} \, \text{robot} \, r \, q) \]

In this case, an individual robot is a robot with a specific serial number. For instance, robot number 25 is an inhabitant of the following type:

\[ \text{robot}25 : (\Sigma x : \text{robot}, (\text{eq} \, \text{nat} \, (n \, x) \, 25)) \]

If the variable ‘n’ has an interpretation, i.e. if \( n \in \mathcal{S} \), then the agent knows how to observe its serial number if he observes a robot, and he is able to recognise this individual robot.

A variation on the same theme is the use of subtypes. The ‘advantage’ of the use of subtypes is that the agent is not required to be able to give a precise explicit description of the criterion that he uses to identify an individual.

For example, if R2d2 is a robot, and we have: \( \Gamma_A \vdash r2d2 : \text{robot} \) and the agent \( A \) is able to recognise R2d2 as an individual, then this ability can formally be explained as caused by \( A \)’s ability to recognise inhabitants of a particular subtype of robots, i.e the type \( t_{r2d2} \). This particular type has an interpretation for the agent. Also, the type is known to have exactly one inhabitant, which is of course equal to \( r2d2 \). If we want to express this properly, we have to write:

\[ t_{r2d2} < \text{robot} \]

\[ r2d2 : t_{r2d2} \]

\[ [x : t_{r2d2}] \Rightarrow (\text{uniq} \, x) : (\text{eq} \, t_{r2d2} \, x \, r2d2) \]

It is assumed here that \( t_{r2d2} \in \mathcal{S} \). Of course, as our mechanism to deal with subtypes is based on \( \Sigma \)-types, this example only differs from the one using serial numbers in one aspect: the criterion that the agent uses to recognise R2d2 is not made explicit.

Making observations

As we have defined the set of observable types, and understand how the agent might recognise individuals, we are ready to look at a few examples. Consider the context \( \Gamma_A \). An example of an observable type is now the following:

\( (\Sigma r : \text{robot}.(\text{red} \, r)) \)
3.6. **Intensionality**

Successfully observing an inhabitant of this type shows the agent that there is a red robot. This is something that the agent already knew, as he knows that R2d2 is red. Is the type: \((\text{red } r2d2)\) itself also observable? At present, this is *not* the case, as the variable \(r2d2\) is not included in the set of interpretable primitives 3. If \(A\) could identify R2d2 by one of the mechanisms that we discussed, i.e. through its serial number, or due to subtyping, he might see that this robot is red, for instance, by observing the type:

\[(\Sigma r : t_{r2d2}(\text{red } r))\]

Another observable type is the following:

\[(\Sigma r : \text{robot}(\text{white } r))\]

Will the agent \(A\) ever observe an inhabitant of this type? We can only guess.

**Other possibilities**

The account of observation that is presented here offers a host of interesting possibilities. In particular, it allows us to distinguish between agents who have different observational abilities. One can for instance construct an agent who is ‘colour blind’ if one robs him of the ability to observe one of the spectral intensities. For instance, we may take away the agent’s ability to compute the spectral intensity of lower wavelengths. Formally this means that we now have a set of interpretable primitives \(3’\) where \(3’ = 3 \setminus \{r\}\). Note that the affected agent can still know everything about colours, without, however, being able to observe the difference between some of them. If such an agent were asked to indentify a red tile, for instance, he would have to *infer* that a given tile is red, combining the facts that he *knows* about the world with the result of such observations as he still can make.

### 3.6 Intensionality

We have shown so far how type systems can be used to construct information states for an agent that are computational, partial, extensible, and semantically grounded in the personal observations of the agent.

We now show that information states which are constructed in this way are intensional, in the sense that:

- They enable the agent to treat non-existent entities, like unicorns, dragons or square circles in a meaningful way.
• They enable the agent to deal with co-referential expressions, enabling him to distinguish between them.

• They will cause the agent to exhibit a kind of 'confusion' when confronted with ambiguous stimuli, like the famous Wittgensteinian 'duck-rabbit'.

3.6.1 Nonexistent objects

It is important that non-existent objects have meaning. For how can one assert that there are no unicorns, if, by this sheer fact alone, the utterance would be meaningless? How can one assert that there is no greatest prime, if, by this fact alone, the assertion is gibberish? Evidently, a satisfactory theory of meaning must be able to deal with non-existent objects.

We can make some progress in these matters along the lines proposed in this thesis. One's ability to recognise a unicorn is in no way dependent on the existence of unicorns. Nor is recognising a unicorn in any way the same thing as recognising a dragon, or a centaur. Moreover, as we already mentioned in the last chapter, whether an agent is able to recognise inhabitants of a type does not depend on the possible existence of an inhabitant of this type from a logical point of view.

To show all this with more rigour, we will give a historical mathematical example. Consider the famous four-colour conjecture, that boils down to the assertion that any finite planar map can be coloured with no more than four distinct colours in such a way that no two neighbouring countries get the same colour. A counterexample to this conjecture would be a finite planar map that needs five different colours to colour it. One is perfectly able to recognise such a map, even if there is no such map; there is nothing mysterious about this ability. In fact, it is also possible to program a computer to recognise such a map. As a matter of fact, existing proofs of this conjecture [Appel & Haken, 1977], [Appel & Haken, 1989] have been completed with the help of a computer that essentially was programmed to do precisely this. It was first shown that the problem can be reduced to a finite but rather large number of cases in graph theory. As these cases where too numerous and complex to be inspected by a human, a computer was programmed to scan the resulting list, which was, at least on an abstract level, a list of possible counterexamples. As the reader probably knows, no case that could lead to a counterexample was found on the list, and the four-colour conjecture is now a theorem. We think it is indisputable that one can say: 'This computer was searching this list for — a suitable abstraction of — a five-colour planar map.' Most mathematicians are prepared to judge this sentence as being true. Not because there are possible worlds in which there

\footnote{For a short history see: [Holton & Purcell, 1979]}
are five-colour planar maps, but because they understand that the computer was programmed to recognise such a map.

3.6.2 Coreferent expressions

An other interesting problem is that of coreferent expressions. This was already noticed by Frege [Frege, 1892], when he presented his famous morning star paradox: Though the morning star and the evening star refer to the same object, i.e. the planet Venus, the assertions: ‘The morning-star is the morning-star’ and ‘The morning-star is the evening-star’ cannot be said to have the same meaning. One of these assertions is a tautology and the other is a contingent truth. So there must be an aspect of meaning that is not captured by the reference of a term. This aspect is what Frege calls ‘der Sinn’, a word mostly translated as ‘sense’. Can this aspect be captured by the account of ‘meaning’ that we advocate here?

Consider the concepts ‘morning star’ and ‘evening star’. It is clear how we could observe those. Recognising the evening star is pretty easy to do, and recognising the morning star is pretty easy, too. To encode these concepts into the knowledge state of an agent we have to introduce two observational types. Now an agent can assume that there are planets Hesperus and Phosphorus that are the inhabitants of these types. (Of course the agent also assumes that the types morning star and evening star have unique inhabitants) The agent may have the following knowledge state;

\[
\begin{align*}
thing & : *, \\
planet & < thing : *, \\
evening & : planet \rightarrow *, \\
morning & : planet \rightarrow *, \\
eq & : \text{Leibniz-equality as given in section 2.4.2} \\
eveningstar & := (\Sigma p : planet. evening p) \\
morningstar & := (\Sigma p : planet. morning p) \\
hesp & : planet,
\end{align*}
\]

\footnote{Zeevat (in chapter 2 of his thesis) remarks that DRT can be used as an interpretation of the phenomenological theory of thought, and can indeed handle Meinongian (i.e. nonexistent) objects. This is not surprising in the light of the correspondence between DRT and type theory that is developed in chapter 6.}
phos : planet,

\[ x : \text{eveningstar} \Rightarrow (u1 \ x) : (\text{eq planet } \pi_1(x) \text{ hesp}) \]

\[ y : \text{morningstar} \Rightarrow (u2 \ y) : (\text{eq planet } \pi_1(y) \text{ phos}) \]

In this knowledge state, the set of interpretable primitives might be:

\[ \mathcal{Z} = \{ \text{thing}, \text{planet}, \text{evening}, \text{morning} \} \]

There is nothing peculiar about the agent’s information state. It is also clearly connected to his observations. When the agent sees the ‘evening-star’, he can identify Hesperus. When he sees the morning-star, he identifies Phosphorus. How does this knowledge state fare with respect to Frege’s problem? The agent knows: ‘Phosphorus is Phosphorus’. But the agent does not know that Hesperus is Phosphorus. Later the agent might learn this. When the agent learns that the morningstar and the eveningstar actually are different aspects of the same planet, he acquires new knowledge. This knowledge can be simply expressed as follows:

\[ c : (\text{eq planet hesp phos}) \]

Or, alternatively, as:

\[ d : (\Pi x : \text{morningstar}.(\Pi y : \text{eveningstar}.(\text{eq planet } \pi_1(x) \pi_1(y)))) \]

The interesting consequences of this new fact may be numerous. If the agent has other information, e.g. that Phosphorus is inhabited, or Phosphorus rotates, then he now also knows that Hesperus is inhabited or rotates.

This example shows that the intensional aspect of meaning which Frege calls ‘der Sinn’ and which in most referential theories can only be captured at the cost of introducing possible worlds, can also be understood more directly in terms of types, if these are semantically grounded by observational criteria as presented here.

### 3.6.3 Ambiguous stimuli

In real life, recognition is not always infallible. It is even conceivable that the same pattern is recognised in different ways. This may happen intentionally, as we may be using different conceptual schemes to ‘understand’ some part of reality, but it may also happen by accident, because the pattern is ambiguous. A well-known example is the pattern presented by the picture in figure 3.1.
This pattern may give rise either to the recognition of a duck, or to that of a rabbit. Could our account of recognition deal with the situation that arises when such a pattern occurs, and what will happen? Clearly we are dealing with a context in which there are types like 'duck' and 'rabbit'. It is also the case that ducks and rabbits are not the same animals. This can be expressed as follows:

\[
\begin{align*}
\text{animal} &< \text{thing} : *, \\
\text{duck} &< \text{animal} : *, \\
\text{rabbit} &< \text{animal} : *, \\
[x : \text{duck}, y : \text{rabbit}, p : (eq \text{animal } x y)] &\Rightarrow (\text{diff } x y p) : \bot 
\end{align*}
\]

Let \( \mathbb{A} = \{ \text{thing, animal, duck, rabbit} \} \). If both a duck and a rabbit are recognised, the context will be extended to the following one:

\[
\begin{align*}
\text{animal} &< \text{thing} : *, \\
\text{duck} &< \text{animal} : *, \\
\text{rabbit} &< \text{animal} : *, \\
[x : \text{duck}, y : \text{rabbit}, p : (eq \text{animal } x y)] &\Rightarrow (\text{diff } x y p) : \bot \\
a &: \text{rabbit}, \\
b &: \text{duck} 
\end{align*}
\]

Note that 'a' and 'b' indeed have different meanings, for the variable 'a' stands for the recognition of the pattern as a rabbit and contains this inter-
pretation, and similarly for ‘b’ as a duck. So the agent exhibits a kind of confusion. Note that a problem arises if we add the constraint that there is only one animal present. (Since ⊥ cannot now be derived.) As the agent is not able to scrutinise his observational abilities as such, he is not able to solve this problem. He can only signal it.

3.7 Knowledge state dynamics

Having investigated various properties of a knowledge state based on type theory, it may be of interest to consider the ways in which such a knowledge state might develop dynamically. Of course, this question opens up a vast subject, related to various areas of the agent’s cognitive development, which includes learning, both of concepts and of facts, reasoning, belief revision, and observation. We cannot possibly cover all these subjects; instead, our goal is rather modest: we want to show that it is possible to make crude, but computational models for some of these processes, that fit reasonably well in the computational view of knowledge that we present in this chapter. In particular, we concern ourselves with the dynamics of reasoning, observation, and belief revision.

To deal with the dynamics of knowledge states in terms of the dynamics of books, we first consider the dynamics of TT books. We begin by defining three basic transformations that can be applied to books:

- A book can be extended with a new declaration.
- A book can be extended with a new definition.
- A declaration in a book can be retracted.

These three transformations are defined as follows:

**Definition 6** Let \( \Gamma \) be a book. \( \Gamma \) can be extended with a new declaration \( x : T \) to form the book \( \Gamma', x : T \) under the following condition: \( \Gamma \vdash T : s \), where \( s \) is a sort and \( x \) is a fresh variable.

**Definition 7** Let \( \Gamma \) be a book. \( \Gamma \) can be extended with a new definition \( x := E : T \) to form the book \( \Gamma, x := E : T \) under the following conditions: \( \Gamma \vdash E : T \) and \( x \) is a fresh variable.

**Definition 8** Let \( \Gamma \) where \( \Gamma = \Gamma_1, x : T, \Gamma_2 \) be a book. The declaration \( x : T \) can be retracted from \( \Gamma \), leading to a new context \( \Gamma' \) where \( \Gamma' = \Gamma_1, \Gamma_2', \) if \( \Gamma_2' \) is the context that we get by retracting from \( \Gamma_2 \) all declarations \( y : T \) that contain a free occurrence of the variable \( x \) in \( T \), and removing all definitions that reduce to expressions that contain a free occurrence of \( x \).
3.7. KNOWLEDGE STATE DYNAMICS

3.7.1 Resource-bounded reasoning

We have seen that an agent’s criterion to determine what he knows is his ability to construct a realisation for an extending segment that expresses this knowledge within his knowledge state. This criterion for an agent’s knowledge has the drawback that it is not decidable. A computational agent who wants to decide whether he knows a fact expressed by some extending segment ‘Δ’ has to replace this criterion by some decidable approximation. The quality of such an approximation obviously depends on the computational resources that the agent is willing to spend. To denote the approximation of ‘∈’ using resource limit ‘n’, we write: ‘∈ₙ’. It is now reasonable to assume that a ‘computational’ approximation of knowledge is then given by the following:

**Definition 9** An agent A knows a fact expressed by some type T iff he can construct an inhabitant for this type T, using his resource-bounded reasoning strategy. Formally:

\[ \exists x. \Gamma_A \vdash _n x : T \]

If we accept this idea, it allows us to model how ‘thinking’ about a subject may increase an agent’s information about that subject from a computational point of view. To see this, we investigate the epistemic situation of the agent in somewhat more detail. We can distinguish at least three different types of information that the agent has:

- **Explicit knowledge**, i.e. everything which is already directly represented in the agent’s knowledge state, in the form of definitions or declarations:

  \[ \exists x. x : T \in \Gamma \]

- **Accessible knowledge**, i.e. everything which can be shown to follow from the explicit information, given the limitations of the agent’s particular deductive strategy with resource bound \( n \):

  \[ \exists x. \Gamma \vdash _n x : T \]

- **Implicit information**, i.e. everything which logically follows from the knowledge state:

  \[ \exists x. \Gamma \vdash x : T \]

Making these distinctions, we see that an agent can increase his accessible knowledge in situations where his implicit knowledge does not change. The
most straightforward, but also rather uninteresting way to achieve this, is to increase the computational resources of the agent, as parameterised by the number \( n \). Another, simple, cheap, and rather powerful mechanism to also achieve this is the recording of intermediate results. As has been discussed in chapter 2, such a mechanism is provided by definitions. We can capture this important dynamic aspect of an agent's knowledge state in the following procedure:

**Procedure 1 (conclusion)** An agent with context \( \Gamma \) who concludes that \( P \), extends his book \( \Gamma \) with a new definition \( x := E : P \) to form a context \( \Gamma, x := E : P \). This action is subject to the condition that \( \Gamma \vdash E : P \).

If the agent draws a conclusion, accessible information is made explicit. Note that the agent does not have to restrict himself to types that correspond to propositions, but can also record how he has constructed inhabitants of other types, thereby making the existence of certain objects or even predicates explicit. If the agent does this, his implicit information will not change at all, but his accessible information will grow, given a suitably defined deductive strategy\(^9\), which is what matters practically. When this process is repeated, it can lead to a type of progressive reasoning that jumps from one idea to the next one, in order to construct a coherent argument\(^10\). In this way the information state can be subjected to transformations in which the explicit information is *growing*, and growing in a particular direction, as information is promoted from accessible to explicit information.

### 3.7.2 Observation

**Definition 10 (observe)** An agent observes that \( T \) if \( T \) is an observable type and the method \( M_T \) returns successfully.

Again, it is useful if the agent can record and remember his observations. To enable him to do so, we define the following mechanism:

**Procedure 2 (observation)** An agent who observes that \( P \), extends his context \( \Gamma \) with a new declaration \( x : P \), where \( x \) is a fresh inhabitant of the type \( P \).

In practice, it is often important that reasoning and observation are combined. How can this be modelled? Suppose, an agent is trying to find a realisation for a segment \( \Delta \), where \( \Delta \) is \( x_1 : T_1, x_2 : T_2 \ldots x_n : T_n \). To do so, he can proceed in two ways;

---

\(^9\)Such a strategy will typically be some ‘bounded depth’ strategy, in which we have: \( x : P \in \Gamma \Rightarrow \Gamma \vdash_0 x : P \).

\(^10\)Of course, one needs some creativity in order to identify the right stepping stones. This creativity however, does not have to reside in the agent himself, as is obvious in Socratic dialogue.
3.7. KNOWLEDGE STATE DYNAMICS

- The agent may try to construct, within his knowledge state, a term (say $E_1$) that is an inhabitant of $T_1$. If he succeeds in doing so, he must, to finish his task, go on to try to construct a realisation for the segment $\Delta'$, where $\Delta'$ is $(x_2 : T_2 ... x_n : T_n)[E_1/x_1]$

- The agent may try to observe an inhabitant $y$ of $T_1$, record the fact that he has seen this $y$ (by the simple procedure given above) and go on to try to construct a realisation for the segment $\Delta'$, where $\Delta'$ is $(x_2 : T_2 ... x_n : T_n)[y/x_1]$.

In fact, there is nothing that prevents the agent from employing a combination of both strategies, and use both observation and reasoning to find an inhabitant for a given type. In this way, the agent can combine his observational and reasoning abilities. Note that this has the curious and rather realistic side effect that the recorded observations will remain in the knowledge state of the agent, even if the observed ‘candidates’ eventually do not lead to the successful construction of a realisation.

3.7.3 Revision of beliefs

The fact that an agent is not able to oversee all the consequences of his current knowledge has another important consequence. His knowledge may be unreliable, or even contain an implicit contradiction, and the agent may fail to notice this for a long time. However, when the agent has drawn a new conclusion, making accessible information explicit, or when he has made a new observation, he may end up in an information state where a contradiction is now accessible.

This means that the agent’s knowledge state contains some assumption that is false. The agent can only set this situation right if he rejects one of the assumptions that has led to the contradiction. To do this, he must contract his information state by retracting this assumption, together with all the consequences of this assumption. I will now briefly indicate how an agent who reasons in TT might try to do this. For a more thorough treatment, the interested reader is referred to [Borghuis & Nederpelt, 2000].

In order to contract an information state that contains an explicit contradiction, one first has to identify the culprit. In general, when we have proved a proposition $P$ we have constructed an inhabitant $E$ of $P$, that is a formal representation of this proof. The object $E$ contains all information about the way in which $P$ has been proved. More precisely, each free variable that occurs in $E$ indicates that the proof in question depends on the object in the book that this variable refers to. This variable may refer either to a declaration or to a definition. In the latter case, one can (recursively) expand this definition in order to get all the declarations on which the proof ultimately depends.
3.8 Conclusion

We have shown how one can employ type theory to represent the knowledge state of an agent. The type-theoretical knowledge state of the agent is related to an external reality through the agent’s ability to recognise inhabitants of certain types observationally. The semantically grounded knowledge states that arise in this way have several interesting properties:

- They equip an agent with a subjective, computational account of meaning that the agent himself can apply, and which is intrinsically intensional.

- They endorse a proof-theoretic view on knowledge, that can easily be translated into a decidable, computational, and resource-bounded notion of knowledge.

- They are extensible and enable an agent who is roaming his environment to gather new information that is meaningful within his knowledge state.

- They offer a practical and foundational approach to the problem of belief revision, in which the problem can also be investigated in its entire scope, taking both reason and observation into account.

A further interesting aspect of type-theoretical knowledge states is that they are not only semantically grounded, but that they are semantically grounded in an explicit way, enabling us to tell whether or not certain information is in principle observable to certain agents, thereby enabling us to distinguish agents who have different observational abilities.

---

11 In our view the identification of the culprit should not take place on the basis of pure reason alone. It seems likely that one should also use one’s ability to observe and to communicate whenever one is confronted with conflicting information. For a different viewpoint, see [Gärdenfors, 1988].
3.8. CONCLUSION

3.8.1 Going on

The grounded knowledge states that we considered so far have several important limitations. Among other things, it is not clear how agents who have such knowledge states are able to learn concepts, how they can communicate with each other, and how their knowledge can be adapted to changes that take place in the outside world.

In this thesis, we will not deal with the learning of concepts, and only briefly comment on it in section 8.2.2. Communication will be discussed in chapters 6 and 7. First, we will concentrate on questions that relate to time and change. How can an agent have consistent knowledge about a changing world, and reason about the events that take place in it? Our answer to this question will be given in the next two chapters.

In chapter 4 we first present a consistent, physically inspired ontology of a changing reality, in which both events and objects figure. Formalising this ontology in predicate logic, we then show how it is possible to reason about a changing world within this ontology.

In chapter 5 we will then consider how an agent who has the kind of knowledge state and the kind of semantic grounding presented here, has to be adapted to develop a consistent picture of a changing world, a picture in which the agent’s knowledge about the history of that world can grow over time.
Chapter 4

Modelling dynamic objects

You cannot step twice into the same rivers,
for fresh waters are ever flowing in upon you.
Heraclitus

In this chapter we construct an ontology of objects and events that is based on ‘physical’ intuitions. In this ontology the changing ‘universe’ is considered as a decomposable structure that consist of different parts. Objects are parts of the universe, and have a time-dependent state. All events are supposed to be, at least in principle, understandable in terms of state changes. This ontology draws a distinction between the identity of objects, their state and their changing circumstances.

We formalise this ontology and develop a method for reasoning about changing objects in a formal framework. In this method, which explains change in terms of state changes, the time dependence of propositions is moved out of the predicates into the objects themselves. It turns out that such a treatment has several advantages over more conventional approaches:

- Because objects are regarded as complex structures that can be decomposed into parts, events can be localised within specific parts of objects. An event only affects a given part of an object if the event changes the state of this part.

- As the structure of objects is made explicit, we can formalise the intuitive distinction between ‘intrinsic’ predicates that depend only on the state of an object, and ‘context-dependent’ predicates that depend on the whole ‘situation’ of an object. In fact, the way in which properties of objects depend on the state of their parts can be formalised.

- From the state-changing effects of an event, and the way in which properties depend on the parts of objects, one can deduce whether, how and why the properties of objects are affected by certain events.
• All facts, including those about different moments in time, can be combined within one and the same monotonic theory.

To illustrate the power of the method, we also present a number of examples involving robots that have to paint a room. The examples deal with increasingly complex situations. In each examples detail is added to the previous one. This is possible because the method is monotonous, and allows reasoning about change in worlds that are only partially specified.

4.1 The problem of change

Humans reason routinely about the way in which actions and events change the world around them. Given enough information, they perform reasoning tasks that involve their future, as when getting groceries, designing a garden, or programming a computer. In fact, they are even able to analyse their past and future in causal terms, as when diagnosing and repairing a broken toy, or when investigating (in retrospect) the cause of an aircraft accident. Apparently, humans have an intuitive understanding of the nature of change, and can employ this understanding to reason successfully about the interlocking effects of different events at different moments in time.

Attempts to formalise this understanding run into severe difficulties. From a technical viewpoint, the main problem seems to be that facts about objects at different moments in time tend to be incompatible, leading to inconsistencies when they are combined. A door which is now open, may soon be closed. And I, who am now bigger than my son, may one day be smaller than he is. Thus (in the standard view) the facts about the world are altered, and facts that are now true, may later become false. Therefore, it seems impossible to combine all the facts about the world within a single theory.

4.1.1 Conventional approaches

As we have seen in chapter 1, various approaches have been proposed to circumvent this difficulty. Most of these approaches have in common that they treat change as logical change, i.e. in such a way that the transitions between worlds or situations involves some kind of theory change. To be able to do this, these approaches use extensions of conventional logic, such as modal operators or non-monotonic update rules.

Is this really necessary? Of course, it is unavoidable that theory change should take place when an agent is confronted with surprises. It seems only fair that our world view has to be adjusted if our expectations turn out to be unreliable. In these cases an agent is forced to adjust his world view, and is involved in a complex epistemic process, possibly involving belief revision or truth maintenance. But it is clearly not the case that changes are always
surprising. In real life, changes occur routinely, and our knowledge of the present situation will often even enable us to predict what changes are about to take place. It seems a bit odd, to say the least, that certain facts in a theory which correctly predicts a certain event, should have to be retracted because this event does indeed occur! Nevertheless, this is precisely what most standard approaches to change want us to believe. For this reason, we believe that something is fundamentally wrong with these approaches. Change itself is not illogical, nor is it paradoxical. In fact, the actual world that we live in, is a changing one. Given that this world exists, it follows that change can be modelled in a consistent way. So, barring those — important — problems that are related to surprises or lack of information, and which are of an epistemic nature, it simply must be possible, at least in principle, to formalise change within a monotonic logical setting.

We will show in this chapter that one can indeed develop a formalisation of time and change in which information about different moments in time can be consistently combined. This formalisation is quite straightforward, but it is based on a structural view of change, which is rooted in an unfamiliar ontology. This alternative ontology allows us to reason about change in standard logic, thus narrowing the gap between logical - i.e. declarative- and procedural - i.e. state based - descriptions, and provides interesting insights, showing us how the time dependence of predicates can be rooted in structural changes of the world.

There are many problems that our method of formalisation does not address. For instance, real-world systems are so complex and rich in detail that they defy attempts at formalisation. This is simply a fact of life, and the approach that we advocate does little to change it. Also, there is more to the understanding of a dynamic situation than the ability to produce formal proofs about it. In particular, there are important judgements about probabilities, or about the causes of certain events, that also play a role. We will not deal with any of these issues. But we will show, using a small but extensible example, that the view on object dynamics as advocated here can be used to reason formally about the behaviour of abstract dynamic models within a monotonic framework, such as predicate logic. The approach that we propose is particularly suited to gain a formal understanding of the dynamics of sizeable, well-defined artifacts like parts of cyberspace.

4.1.2 The root of the problem

Before introducing this alternative view, we first emphasise the central ontological assumption that lies at the heart of all the conventional formalisations of change, for it is this assumption, that we will challenge.

In all the three approaches that we have seen in chapter 1, it is tacitly assumed, that the 'common sense objects' which we all know from experience
and about whose behaviour one tries to reason, correspond in a straightforward way to logical terms and do not admit further temporal analysis. Let’s take a simple example and show the — unwanted — implications of this assumption. Consider the car which John bought ten years ago, and which is now rather time-worn. In order to formalise its history one typically introduces a constant, say: ‘c’ which refers to this car. The formalism contains propositions about this car, which are expressed as predicates with the argument ‘c’. As ten years ago, in 1991, the car was brand new, there was evidence for the proposition: shining(c). Today the car is rather ramshackle, and there is ample evidence for the proposition: blotched(c). The two facts cannot be stored within one single consistent theory, and special measures have to be taken to combine them within one framework.

In this example, it is obvious that the way in which the changing reality has been coded in the theory has two important consequences:

- On the one hand it enforces changes to affect the extension of the predicates. For instance, the predicate ‘shining’ contains the object ‘c’ in 1991, but it no longer does so now. For the predicate ‘blotched’ it is just the other way round. In a way, this is peculiar: do we have to conclude that the meaning of predicates somehow depends on time?

- On the other hand, at all moments in time the formalism refers to this changing car using one fixed logical expression, the term ‘c’. Thus, whereas the predicates are forced to change, the car ‘c’ itself seems miraculously static (at least from a formal viewpoint) through all those years. It somehow seems that this is just the wrong way round: the objects should change, not the predicates.

In what sense is the car ‘c’ the same car it was ten years ago? Is this car as it is right now indistinguishable from the car as it was when John bought it? That is rather unlikely. One is in Vegas, and the other was in Boston. One of them is blotched, while the other one was smooth and shining. One is, and the other was. Given all these facts, it seems questionable whether it is profitable to code a changing reality in a framework that uses a single logical term to refer to a given object throughout its entire history. In any framework that does this, the different ‘beings in time’ or ‘time slices’ of an object are logically indistinguishable. But when we are reasoning about changes, or think about the course of time, we are very well able to differentiate between a given object as it was some time ago, and the same object as it is right now. The need to be able to make such distinctions is increased by the fact that the changing objects that inhabit the universe around us do not resemble the point-like inhabitants of mathematical models at all. They have a complex physical constitution, that is changing over time. Indeed, there is a natural
and obvious distinction between the blotched car John possesses today, and the shining one he bought ten years ago.

There is of course a sense in which both these cars are 'the same', but we maintain that there is an equally important sense in which they are different. It follows that a reasoning framework that is used to express what happens when objects are changing, must be able to mirror this distinction, if it is to avoid unwanted paradoxes. In this chapter we show how these paradoxes can be avoided by coding the changing universe using conventional logic in such a way that the distinctions between the different time slices of 'one and the same' object are respected by the formalism. The proposed encoding of a changing reality rests on a structurally oriented view of change, in which events and objects both figure as first-class citizens, and in which we emphasize that objects are not point-like static entities, but that they are decomposable structures, that undergo local state changes.

In the next section we will first outline this view, and argue why we believe it is advantageous. We will then introduce the various ontological categories that play a role within such a view. The reader will encounter various familiar entities like moments, events, and historical objects that persist through time, but also some less familiar ones, like structures, selectors of substructures, and time slices. Clarifying the relations between these various entities, it is possible to sketch a consistent picture of a changing universe that can be cast into a mathematical mould, thus yielding a formal model. This model reflects a structural view on a changing universe, where changes are located within the different structures that make up this universe.

Axiomatising such a model within a monotonic and declarative reasoning framework, we will show that it is possible to use standard logics to reason about change.

4.2 A structural view of change

We want to develop an ontological picture that does justice to the intuition that objects always are in a certain state and that the truth of propositions about objects ultimately depends on these states. In this view, the truth value of propositions only changes as a consequence of state changes within certain objects. The idea is that one understands the universe and the way in which it changes because one is decomposing it in terms of changing structures. If an event happens, the state of some structures will change, but most structures remain unaffected. To understands certain events well implies that one is aware which structures will change due to the events.

In order to reason or to talk about changing structures one has to organise and categorise them, and to know which structures are part of other structures. To do so, we assume that there is a largest changing structure, the
universe. This universe can be divided into parts. This division is repetitive: parts can again have parts. In this way one can form an ordering\(^1\) of decomposable structures, at the top of which sits the universe as a whole. Certain parts of the universe, or structures, are distinguished in some way, and it is both possible and useful to track them in the course of history. These parts correspond to what one calls the ‘objects’ in the universe. About these objects, one can assert certain propositions. In our analysis, the truth value of a proposition about some object ultimately depends on the current state of the structures that constitute the object and its surroundings, and can change, because the structures on which the proposition depends can undergo state changes.

Indeed, in order to change the truth value of a proposition, the state of certain parts of the universe has to change. To analyse the dynamics of a certain proposition one needs to know which parts of the universe must change to change the truth value of this proposition. A precise understanding of the dynamics of a time-dependent proposition can be achieved if this proposition can be expressed as a condition on the states of objects.

To gain such an understanding we have to unravel the way in which the changeable properties of an object depend on the changing states of the different parts of the universe. This includes the dependency of the properties of an object on the state of the object itself. To clarify this, one first has to consider the different views that one can have on objects:

- One view is that objects are *enduring* entities. This is often considered to be obvious, and it is this view that is conventionally stressed, as we have seen in section 4.1.2. Abstracting over the consecutive changes that an object undergoes, one perceives an object as having some unchanging identity.

- Another view is a *local* one. It focuses on the ‘local’ aspects of an object, and abstracts both over its identity and over its spatial and temporal context. At each moment in time, the part of the universe that constitutes an object is in a certain local ‘state’. This state can only change if the object is involved in some event. For instance, looking at John’s car right now, its doors are locked, one of its windows is slid open, the engine is not running, its colour is Prussian blue, it is 3.50 meters long, etc. All these things are true by virtue of the current state of that part of the universe that constitutes John’s car, and they do not depend in any way on the spatial or historic context of this car, nor on its historic identity. To change any of these facts, one must change that part of the universe that corresponds to the car. As we

---

\(^1\)The ordering is a partial one.
will see, this local view of things allows us to single out properties and attributes that are 'intrinsic' to the car.

- A third view – which is the most comprehensive one, because it refuses to abstract over anything – takes into account that objects are found in different contextual circumstances at different moments in time.

This view is needed to understand how the propositions that hold for a given object can change, even if the state of this object itself does not change at all. For instance, when we look at John’s car now within its actual surroundings, we see that it is parked between a yellow and a red car, and that it is the biggest blue car in the parking lot. Whereas when we looked at this car an hour ago, it was parked between two white cars, and it was not the biggest blue car in the parking lot. So certain properties of the car have changed, though the state of the car itself is unchanged. These properties are context dependent. We can only understand this on a view that also takes into account the changing circumstances of the car.

Whereas the first two views are familiar, the third one may seem somewhat unorthodox. It raises a problem: Which ontological entity can it be that we proclaim to be “between two white cars”? Clearly, such an assertion refers not to the car ‘c’ as an enduring historic entity. For if it did, the resulting assertion must, from a purely logical point of view, remain true forever, which is hardly the case. But on the other hand, this assertion can also not refer to the state of the structure that constitutes the car. For the assertion may be invalidated by some event, even if this event produces no accompanying state change in the car.

To solve this problem, it will be necessary to introduce a third ontological entity, one that embodies the circumstances of an object at a particular moment. We will do this, and call such ontological entities: time slices. Given some object – i.e. an enduring entity – and a particular moment, we combine the object and the moment, forming some short-lived entity whose circumstances one can judge against the background of the universe. The time slices of objects will intuitively correspond to the incarnations of an object at different moments in time, and it is these slices – as opposed to the states that they happen to be in at a particular moment – that can be burdened with context-sensitive properties. So, in order to deal with the way in which context-dependent properties are changing, we treat the various time slices of an object as ‘first-class citizens’ and accord them a central place in our ontology.

Time slices constitute the most fine-grained level at which objects can be distinguished. If two time slices are equal it follows not only that they have the same state, but also that they are slices of the same object, that
they are taken at the same moment in time, and that they share, in fact, all properties. We will not enter a philosophical debate about the nature of such time slices. For our purposes, it is enough that they can be modelled mathematically, as we will show in section 4.3.1.

The importance of the three different views lies in the fact that an ontology that consistently distinguishes and combines these three views enables us — as we will show — to achieve a formal and more thorough understanding of the way in which the changing properties of objects are rooted in state changes of these (and other) objects.

As we introduce an extra level of 'states' between objects and their properties, our understanding of the way in which the truth values of propositions may change is an indirect one. But it has two important advantages:

- First, it guides us towards a model of a changing universe in which different moments of time are integrated. Such a model not only deepens our understanding, but — when captured logically — also enables us to describe a changing world within a single monotonic logical theory. A description of object dynamics in terms of a single monotonic theory is attractive because it enables us to combine information about different moments in time without further ado.

- Second, it allows for state changes to be located in parts of the universe. Structures are composed of substructures. They can be decomposed into parts, using spatial or other, more abstract, ways of dividing them. Accordingly, the changes that take place in these structures can be confined to parts of objects, and it is possible to have reasonably concise and informative descriptions of the changes that are caused by different kinds of events. Thus, if we are able to relate the truth value of propositions to conditions on changing structures, it is possible to understand whether, why and how an event changes a proposition.

4.2.1 Examples of localisation

In order to reason about the way in which a given event changes a given proposition, one needs to combine two kinds of knowledge: knowledge about the state-changing effects of the event, and knowledge about how the proposition depends on the states of different parts of the universe. If one has sufficient knowledge of both kinds, one can determine whether the event can affect the truth value of the proposition, taking advantage of the fact that events are localised. What this means is, that, for a given event, only certain parts of the universe are affected. In other words, the event has a limited area of structural impact, and leaves most of the universe unchanged. On the other hand, for any given proposition there are specific structures that
must change in order to alter the truth value of the proposition. It is intuitively clear that an event can only change a proposition if there is an overlap between those parts of the universe that are affected by the event, and those parts of the universe on which the truth value of the proposition depends. So, if we can somehow express which structures are and which are not changed by an event, and if we are also able to express on which structures a given proposition does and on which it does not depend, we get a formal grip on the problem, and we may prove that a given event cannot possibly affect a given proposition. This is particularly important, because it means that we do not have to use frame axioms or similar devices to conclude that the given event does not affect this proposition.

At first, all this may seem somewhat far-fetched, but as it happens this type of reasoning is quite natural, and arguments of this kind are routinely used in daily life. However, the underlying reasoning process seems so trivial that it often goes unnoticed, and it seems rather childish to make such reasoning explicit. We will illustrate this with a simple example.

Assume that you are ill. You would be rather surprised if your doctor, after having diagnosed you, would swallow a pill herself to cure your illness. It is obvious to you that such an action would not cure your ailment. But why is this obvious? Because you hold two beliefs:

- That the predicate 'being ill' is in fact nothing but a complex condition on the state of your body. You do not know this condition, and are not able to express it, but you do understand that in order to change the truth value of this predicate 'being ill' as predicated about you, one somehow has to change the state of your body.

- That the 'area of impact' of taking the pill is (a part of) the body of the person that takes the pill, in this case, the doctor\(^2\).

So, except in cases where you happen to be your own doctor, it follows from these beliefs that there is no overlap between the area of impact of the 'pill-taking' and the area of sensitivity of the 'being-ill'. Hence your surprise.

This trivial example shows that there is indeed a certain natural tendency to regard change as essentially localised. It also shows that it is possible to draw viable conclusions from a view that emphasises structure, even in cases where both the structures that are considered and the events that are taking place are extremely complex, and only known to a limited extent.

A structural view cannot only be used to determine whether an event might affect a proposition, but also to calculate whether a given event does affect a certain proposition. To be able to do this we must be able to express

\(^2\)Ignoring the effects on your mental state that are a result of your seeing the doctor taking the pill.
not only which structures are changed by certain events, but also in what way these structures are changed. This can be done if we characterise the actual state of these structures, for instance through a number of ‘attributes’ whose value depends on the state of the structure. If we can indicate how the various events affect the values of these attributes and in what way different propositions of this or other objects depend on the actual values of these attributes, we can determine the effect of the events on the propositions.

To pursue the medical example, a practitioner who has some knowledge about the workings of the human body may be able to relate my illness to a specific condition of specific aspects of my body. For instance, she might diagnose me as having too low blood pressure or too high stomach acidity. To treat this condition, she must have precise knowledge about the area of impact of different medicines, knowing that they act on different aspects of my body: blood pressure, stomach acidity, insulin concentration, etc. Apart from this, she also has to know how the different medicines affect these different aspects; whether a certain medicine increases or decreases blood pressure, and so on. It is clear that a doctor with such knowledge will have good reasons to prescribe certain medication, and can be said to have some insight into why a therapy might work.

When our understanding of the structures and events involved is thorough — as when trying to fathom the workings of a computer program, or some other man-made machinery — we can expect to be able to draw precise conclusions. For instance, take the situation where a string of characters can be changed through certain operations, like adding the character ‘b’, coding the string, removing certain substrings, etc. You may now be able to tell how these operations affect certain aspects of the original string, like its length, its parity, the number of occurrences of the character ‘a’, etc.

In general, the extent of the knowledge about the structure of an object determines the depth of our understanding of the dynamics of an object under certain types of events. To have a deep understanding of the dynamics of a certain object means that one is able, for the propositions of interest, to predict whether they are changed by typical events. It is such an understanding that we wish to model.

## 4.3 Ontology

In this section we will present an ontology that enables us to understand a changing universe from a structural point of view. Before we can investigate the different ontological categories, there is a problem that we have to deal with. It is not possible to construct an ontology that models change without some model of time. Now there are many different ways to model time; in particular time can be modelled as discrete or as continuous, as linear or
4.3. ONTOLOGY

as branching. Though such matters are to a large extent orthogonal to the problems that concern us here, we are forced to make some choices in these matters; what’s more, these choices will have important consequences for the way in which events can be modelled.

To avoid unnecessary complications, which would only distract from the main point, we will choose what is in many respects the simplest option, and will deal only with ordered sequences of discrete moments on a linear time scale. Moreover, we will assume that the number of moments between any two given moments is finite. These assumptions limit the usefulness of the model, as, for instance, the resulting model cannot\footnote{This limitation can probably be avoided at the cost of introducing integrals, as happens for example in the duration calculus [Zhou, 1991].} deal with continuous processes. Nevertheless, the class of problems that we can handle is still quite large. In particular, this choice still allows us to reason about a ‘digital’ universe in which there are only discrete changes, i.e. about cyberspace.

Assuming time to be linear and discrete, and the number of moments between any two moments finite, we are in a position to describe and model certain ontological categories that are in accord with a structural view on change. In this section we will present these categories and integrate them in a total picture, showing how a model can be constructed in accordance with this ontology. We will appeal primarily to the reader’s intuitive understanding, but we will also indicate how the members of the various categories can be modelled mathematically. In this way, our intuitive understanding is sharpened and we are provided with precise mathematical descriptions of the various ontological categories.

4.3.1 A catalogue of categories

The categories that we distinguish are the following:

**Time**

Time consists of moments which are linearly ordered. We assume that the number of moments between any two moments is finite. Mathematically, this corresponds to a timeline that is isomorphic with the set of integers $\mathbb{Z}$.

**Definition 1 (Time)** A timeline is a set $T$ with a linear ordering $<$ that is isomorphic to $\mathbb{Z}$, the set of integers.

**Definition 2 (Moment)** For every timeline $T$, if $t \in T$, then $t$ is a moment.

**State spaces**

To model the momentary state of changing structures we need state spaces.
State spaces are modelled as sets, the elements of which are states. A state space must have at least two elements, since state spaces with only one element can not exhibit any dynamic behaviour. The simplest state space is isomorphic to \( \mathbb{B} \), the set of booleans. The most complex state space that we consider is \( \Phi \), the state space of the entire universe. Between these two extremes there is of course a plethora of other possibilities. To describe the time-dependent aspects of the state of objects, one may conveniently use numerical state spaces like \( \mathbb{R} \) or \( \mathbb{Z} \), multi-dimensional vector spaces, or arbitrary finite sets. In the sequel, we assume a collection \( \Omega \) of state spaces, \( \Omega = \{ S_1, S_2, S_3, \ldots \} \). The state space of the universe, \( \Phi \), is an element of \( \Omega \).

**History**
The universe has a particular history, i.e. at each moment of time the universe is in a certain state. Mathematically, the history of the universe is given by a function \( \phi : (T \rightarrow \Phi) \) from all moments of time to its state space \( \Phi \). This function represents how the state of the universe develops in time. Thus, at each moment \( t \), the state \( \phi(t) \) is the actual state of the universe.

**Definition 3 (History)**
The history of the universe is a function \( \phi : (T \rightarrow \Phi) \).

**Structures**
Structures are parts of the universe. The state of a structure can be computed from the state of the universe. A structure can thus be identified with a mapping from the state space of the universe to another state space. This leads to the following formal definition of a structure.

**Definition 4 (Structure)**
A structure is a mapping \( f : \Phi \rightarrow S_i \), i.e. a function with domain \( \Phi \) and a codomain \( S_i \), which is an element of the collection \( \Omega \) of state spaces.

As we will see in definition 8, structures define a (rather liberal) 'part of' relation that is transitive and reflexive.

**Universe**
The universe itself is also a structure. Its state can be computed by the identity function.

**Definition 5 (Universe)**
The universe is the identity function on \( \Phi \), \( \lambda(x : \Phi) : (\Phi \rightarrow \Phi) \).

**States of structures**
At each moment in time, structures are in a certain state. The state of a
structure can change over time. The state of a structure \( f \) is the result of applying it to the current state of the universe.

**Definition 6 (State)** Let \( f : (\Phi \rightarrow S_1) \) be a structure, then the state of structure \( f \) at moment \( t \) is \( f(\phi(t)) \).

**Selectors**
Structures can have parts, which also are structures and can again have parts. For each part of a structure there is a function that calculates the state of this part from the state of the structure as a whole. These functions, which are mappings from the more complex state space of the whole structure into the simpler state space of the substructure, we will call selectors. Selectors pick information about the state of the substructure out of the state of the whole structure, and can be used to decompose\(^4\) the state space of a complex structure.

**Definition 7 (Selector)** Let \( a : (\Phi \rightarrow S_1) \) and \( b : (\Phi \rightarrow S_2) \) be structures. A selector from \( a \) to \( b \) is a function \( f : (S_1 \rightarrow S_2) \) such that \( b = f \circ a \).

We use the standard notation for function composition: \( f \circ a \) is the function that applies \( f \) to the result of applying \( a \).

**Fact 1** Every structure is a selector from the universe to itself.

**Definition 8 (Part of)** Let \( a : (\Phi \rightarrow S_1) \) and \( b : (\Phi \rightarrow S_2) \) be structures. A structure \( b \) is a part of a structure \( a \) iff there exists a selector from \( a \) to \( b \), i.e., if there exists a function \( f \) such that \( b = f \circ a \), i.e., if \( b \) is the composition of \( f \) and \( a \).

**Fact 2** All structures are parts of the universe.

**Fact 3** The 'part of' relation is transitive and reflexive.

**Objects**
People tend to understand their surroundings in terms of objects, which are supposed to be more or less enduring entities. We model objects as special parts of the universe, 'special' in the sense that they are persistently identified by some agent as 'one and the same object'. Mathematically, the set of all objects is just some subset of the set of all structures.

\(^4\)There is no a priori reason to prefer one specific decomposition of a structure over another. It is often useful to be able to decompose a structure in various ways. In a given case, some decompositions are simply more practical than others. As a rule one should try to decompose structures in such a way that, on the one hand, state changes for common events can be confined to a small number of substructures, but, on the other hand, one is able to identify the kind of event that effects a change in a certain substructure.
**Definition 9 (Object)** Objects are the elements of a designated subset of structures.

Notation: We use $\mathcal{O}$ to designate the set of structures, whose elements are called ‘objects’.

**Events**
Events are discrete and atomic, and take place between two successive moments. Mathematically, events are modelled by pairs of successive moments:

**Definition 10 (Event)** An event is a pair $\langle t, t + 1 \rangle$ of successive moments.

The ordering ‘$<$’ between moments can be extended to an order ‘$<$’ that also includes events, in such a way that the event: $\langle t, t + 1 \rangle$ is found between the moment $t$ and the moment $t + 1$.

**Change**
Events cause changes. These are localised in structures. A structure is changed by some event if the state of the structure at the moment right after the event is different from the state of the structure at the moment right before the event.

**Definition 11 (Change)** An event $\langle t, t + 1 \rangle$ changes a structure $s$ if and only if: $s(\phi(t)) \neq s(\phi(t + 1))$

**Fact 4** Let $e = \langle t, t + 1 \rangle$ be an event that does not change a structure $s$. Let the structure $p$ be an arbitrary part of $s$. Then $e$ does not change $p$ either.

Proof: We have $s(\phi(t)) = s(\phi(t + 1))$. Because $p$ is a part of $s$, there is a function $f$ with: $p = f \circ s$. Therefore: $p(\phi(t)) = f(s(\phi(t))) = f(s(\phi(t + 1))) = p(\phi(t + 1))$.

**Intrinsic properties**
There are many properties of objects that do not depend on the circumstances of the objects, but only depend on the local state of the object. Take the case where you are presented with a car at some particular instant ‘$t$’, and are asked as to whether it is blue or not. In order to answer you only have to consider the actual state of the car at that moment. By comparing the momentary colour of the car — a specific hue of Prussian blue — with prototypical colours you may then judge that the car is blue. The circumstances of the car do not enter into consideration anywhere. This is a rather typical case, which also holds for many other properties of objects, like mass, form, or temperature. The properties which only depend upon the state of an object, we call **intrinsic** properties. Note that properties are structures,
4.3. **ONTOGONY**

selectors from the state space of the objects to \( \mathcal{B} \), the set of booleans. If a property is intrinsic it can only be affected by an event that affects the structure of the object of which it is predicated.

**Definition 12 (Intrinsic Property)** Let \( p \) be a function from \( \Phi \) to \( \mathcal{B} \). Let \( x \) be an object. If \( p \) is a part of \( x \), then \( p \) is an intrinsic property of \( x \).

This definition is justified by the following observation:

**Fact 5** Let \( x \) be a structure and \( P \) a property. If \( P \) is an intrinsic property of \( x \), then it can only change due to an event that also changes \( x \).

**Attributes**

One can define structures whose state spaces are well-defined mathematical values, like vectors or quantities. If such a structure \( A \) is a part of a certain object \( x \), it will be called an attribute of this object. In this case there exists a selector \( s \) that calculates the state of the attribute \( A \) (i.e. its value) from the state of the object \( x \). Attributes are useful, because one can specify the state of an object by giving the values of a number of attributes of this object.

**Fact 6** Let \( x \) be a structure and \( A \) an attribute of \( x \). Then \( A \) can only change due to an event that also changes \( x \).

A special kind of attributes, that we will often encounter in our examples, are attributes whose values are quantities, like numbers, that can be compared. Such attributes are called **quantitative**.

**Definition 13 (Quantitative Attribute)** Let \( A \) be a function from \( \Phi \) to some quantitative state space (like \( \mathbb{R} \) or \( \mathbb{N} \)), i.e. a state space which has a linear ordering \( \prec \). Then \( A \) is a quantitative attribute of a structure \( x \) iff \( A \) is a part of the structure \( x \), i.e. iff there is a function \( f \) with \( A = f \circ x \).

**Time slices**

Objects not only have states, but can also be found in certain circumstances at some given moment. To express this, we allow objects to be combined with a moment to form a **time slice**. From any object one can form a host of different time slices, one for each moment in time. All these time slices are **distinguishable**. A time slice of an object refers to a given object at a given moment. For this reason a time slice can be located precisely in its entire context. Accordingly, time slices can carry dynamic context-dependent properties. Mathematically, a time slice corresponds to a pair:

**Definition 14 (Time slice)** The time slice of an object \( x \) with time \( t \) is the pair \( \langle x, t \rangle \).
As all time slices are unique, they will enable us to make an assertion about the way in which an object is at that time embedded in its temporal or spatial context. In order to refer to the circumstances of a unique object \( x \) at a unique moment \( t_1 \) one combines this object with that moment. The resulting time slice \( \langle x, t_1 \rangle \) may be said to have a certain property ‘\( P \)’, whereas the combination of the same object with another moment, \( \langle x, t_2 \rangle \) may be said not have the property \( P \). This can be interpreted as if the truth value of the property \( P \) which is attributed to an object \( x \) has changed between the moments \( t_1 \) and \( t_2 \). If judgements about objects are understood in this way, it is perfectly understandable how one can make contradictory assertions about the ‘same’ object at different moments in time. This does not lead to inconsistencies, because these judgement refer to time slices of the same object at different moments.

The reader should note that it is important that time slices are not confused with the states of the objects. An object can be in different relations to other objects at moments \( t_1 \) and \( t_2 \), even if its state does not change between \( t_1 \) and \( t_2 \). Indeed, if \( t_1 \neq t_2 \) then one has \( \langle x, t_1 \rangle \neq \langle x, t_2 \rangle \) even if: \( x(\phi(t_1)) = x(\phi(t_2)) \).

### 4.3.2 Looking back

What have we achieved so far? We have given a mathematical model of the various ontological entities that play a role in the structural view on time and change that we advocate. In this model figure time, states, structures, objects, events, attributes, properties and time slices. We have given a criterion for what it means for a particular event to change a certain object. As we can also tell which objects are parts of other objects, this criterion enables us to localise change within objects. Indeed, it is a peculiar feature of the model that it enables us to distinguish between the intrinsic and the more context-sensitive properties (or attributes) of objects.

The given ontology does not require us to specify the precise nature of state spaces or the exact ‘state’ of an object. What is important is that certain properties are intrinsic. To show that this model can provide the clarity that we need to reason about time and change, we go on to formalise this model. For this purpose we will use (many-sorted) predicate logic. This will allow us to reason about the effects of events in a monotonic framework.
4.4 The logical framework

Agents have incomplete\textsuperscript{5} knowledge about the history of their particular 'universe', and would like to derive new interesting facts about this history from the knowledge that they have. To do this, they will have to reason.

To capture the agent's reasoning process formally, we need a logical framework. Such a framework is built around a formal language which allows us to express the knowledge that the agent has about his changing universe, both the knowledge about particulars and the knowledge about general laws.

4.4.1 Representation language: syntax

As representation language we use first-order predicate logic. To deal with the ontological categories distinguished in the last section, it is most effective to use a many-sorted predicate logic, reserving the different sorts for the different ontological categories. More precisely, we use a many-sorted predicate logic with function symbols and equality. In this logic there will be terms that correspond to:

- moments: the elements of time
- events: the state changes of the universe.
- states: the elements of state spaces.
- objects: the structures that change, but which persist in time.
- time slices: the combinations of an object and a moment, at which an object is found in certain circumstances.

The following notational conventions for different sorts of atoms, functions and predicates will be useful.

Atoms

- To refer to moments, the letter 't' is used, possibly with subscripts, e.g. 't\textsubscript{i}'.
- To refer to events, the letters 'e, f' and 'g' are used, possibly with subscripts, e.g. 'e\textsubscript{i}'.
- To refer to states, the letter 's' is used, possible with subscripts, e.g. 's\textsubscript{i}'.

\textsuperscript{5}They lack knowledge about that part of the history of the universe that lies in the future with respect to themselves, and in many cases also lack important knowledge about the past, as when investigating a murder case or an accident.
CHAPTER 4. MODELLING DYNAMIC OBJECTS

- To refer to objects, any other letter is used, possibly with subscripts, i.e. ‘a, b, c, l, r,...’

Note that there are no atomic terms that refer to time slices.

Functions

Functions that occur in terms are either selectors, i.e. functions from one state space to another, or they are special functions. There are four special functions, for which we use the following notation:

- There is a function which, when given an object \( b \) and a moment in time \( t \), yields the time slice of \( b \) at the moment \( t \). This time slice is written as: ‘\( b \& t \)’.

- There is a function which, when given some time slice \( x \), yields the state of this time slice. The state of time slice \( x \) is written as: ‘\([x]\)’.

- There is a function which, applied to an event \( e \), yields the moment just before the event. The moment just before an event \( e \) is written as: ‘\( \beta(e) \)’

- There is a function which, applied to an event \( e \), yields the moment just after the event. The moment just after an event \( e \) is written as: ‘\( \alpha(e) \)’

Apart from the special functions, all functions that may occur are selectors.

Predicates

To be well-suited for our purposes, the formalism must have three special predicates:

- The predicate ‘\( = \)’ that stands for (Leibniz) equality.

- The predicate ‘\( < \)’ that stands for a linear order between elements of certain (quantitative) state spaces.

- The predicate ‘\( \prec \)’ that stands for a linear ordering between events and moments.

The logic that the agent is going to use is an ordinary many-sorted predicate logic with function symbols, equality, the quantifiers ‘\( \forall \)’ and ‘\( \exists \)’ and the standard connectives ‘\( \wedge, \neg, \vee, \rightarrow \)’, all with their conventional meanings. Quantifiers are restricted over specific sorts. All quantifiers quantify over: moments, events, states or objects. There are no quantifiers over time slices. As there exist no other time slices than those that can be formed by combining an object and a moment using ‘\&’, there is no need for a separate quantifier over time slices.
4.4. THE LOGICAL FRAMEWORK

4.4.2 Representation language: semantics

To understand how the expressions in the formalism are related to a mathematical model of a changing universe, we first construct this mathematical model, in accordance with the ontology presented in section on ontol. A model $M$ of a changing universe is a tuple:

$$ M = < \Phi, \Omega, \mathcal{O}, T, \phi > $$

where:

- $\Phi$ is a set, the state space of the universe. The elements of $\Phi$ are called ‘states’.
- $\Omega$ is a collection of sets with two or more elements, also containing $\Phi$. The elements of $\Omega$ are called ‘state spaces’.
- $\mathcal{O}$ is a subset of the set of structures, i.e. a subset of the set of functions from $\Phi$ to some state space $S_i$, where $S_i \in \Omega$. $\mathcal{O}$ is the set of all ‘objects’.
- $T$ is a timeline, i.e. a linearly ordered set, isomorphic to the set of integers. The elements of $T$ are called ‘moments’.
- $\phi$ is a function from $T$ to $\Phi$. $\phi$ is the state of the universe as a function of time, i.e. a structural history of the universe.

The language $L$ whose semantics we want to formulate, is a form of many-sorted predicate logic, enriched with a few special functions. We first give the meaning of atoms, then the meaning of functions, and eventually the meaning of the special predicates, following the notation conventions outlined above.

Atoms

- Terms $t_i$ refer to a moment, i.e. the meaning of a term $t_i$ is always an element of $T$: $\llbracket t_i \rrbracket \in T$.
- Terms $s_i$ refer to states, i.e. the meaning of a term $s_i$ is an element of some state space $S$, where: $S \in \Omega$.
- Terms $e_i$ (or $f_i$ and $g_i$) refer to events, i.e. there is a $t \in T$ such that the meaning of $e_i$ is the pair $\langle t, t + 1 \rangle$.
- Terms like $x$, $y$ or $r$ refer to objects, i.e. the meaning of a term $x$ is an element of $\mathcal{O}$: $\llbracket x \rrbracket \in \mathcal{O}$.
Functions

- If the term \( t \) refers to a moment and the term \( x \) to an object then the meaning of the term \( x \& t \) is a time slice, with: \( \llbracket x \& t \rrbracket = \langle \llbracket x \rrbracket, \llbracket t \rrbracket \rangle \).

- If \( x \& t \) is a time slice, then the meaning of the term \( [x \& t] \) is the state of this time slice: \( \llbracket [x \& t] \rrbracket = [x](\phi(\llbracket t \rrbracket)) \).

- If \( e \) is an event with \( \llbracket e \rrbracket = \langle t, t + 1 \rangle \), then the meaning of the term \( \beta(e) \) is the moment at the beginning of the event \( e \) : \( \llbracket \beta(e) \rrbracket = t \).

- If \( e \) is an event with \( \llbracket e \rrbracket = \langle t, t + 1 \rangle \), then the meaning of the term \( \alpha(e) \) is the moment at the end of the event \( e \) : \( \llbracket \alpha(e) \rrbracket = t + 1 \).

- If \( sel \) is a function symbol that does not denote one of the previous special functions, then \( sel \) denotes a selector from a state space \( S_1 \) to a state space \( S_2 \), with \( S_1 \in \Omega \) and \( S_2 \in \Omega \).

Predicates

- The meaning of ‘\( = \)’ is (Leibniz) equality.

- The meaning of ‘\( < \)’ is the conventional linear order between numbers.

- The meaning of ‘\( \prec \)’ is the ordering on moments and events as given by:
  
  - For two moments \( t_1 \) and \( t_2 \):
    \[ [t_1 \prec t_2] = [t_1] < [t_2] \]

  - For a moment \( t_1 \) and an event \( e \) with: \( \llbracket e \rrbracket = \langle t, t + 1 \rangle \)
    \[ [t_1 \prec e] = [t_1] < t + 1 \]
    \[ [e \prec t_1] = t < [t_1] \]

  - For two events \( e_1, e_2 \) with \( \llbracket e_1 \rrbracket = \langle t, t + 1 \rangle \) and \( \llbracket e_2 \rrbracket = \langle t', t' + 1 \rangle \):
    \[ [e_1 \prec e_2] = t < t' \]

4.5 How to reason

An agent which reasons about the history of a universe must have many different kinds of knowledge at his disposal.
4.5. HOW TO REASON

- He has to know how the different objects are structured, and he has to have knowledge about the way in which time-dependent properties and predicates of objects can be reduced to conditions on the state of objects. This is structural knowledge.

- He has to have knowledge about the different kinds of events that can happen in the universe, and must know something about the state-changing effects of the different events, and also about the conditions under which such events might occur. This is knowledge about events.

- He has to have knowledge about the way in which moments and events are ordered in general. This is knowledge about time.

- He has to have knowledge about the current state of the universe, or about a previous state, or about certain events that have taken place. This is knowledge about particulars.

Combining these kinds of knowledge, linking information about different moments in time, the agent may draw conclusions about the possible courses of history.

4.5.1 Knowledge about structure

How can an agent express facts about the universe in the language we have defined? Consider the fact that a car ‘c’ is ‘blue’ at some moment $t_1$. This is a judgement about the time slice of this car at this moment, and is expressed as follows: $\text{blue}(c&t_1)$, where blue is a predicate over time slices. As such, this judgement is not yet very informative.

However, one of the things that the agent might also know, is that ‘blue-ness’ is a property that can be reduced to a condition that depends solely on the local state of an object. This can now be expressed:

$$\forall x, t. \text{blue}(x&t) \leftrightarrow bl([x&t])$$

(4.1)

This equation asserts that a time slice ‘$x&t$’ is blue if and only if the state of this time slice observes a certain condition. It reduces the predicate ‘blue’ over time slices to a predicate ‘$bl$’ over states. This is important because equation 4.1 imposes a strong constraint on the dynamics of the property ‘blue’: it expresses that ‘being blue’ is an intrinsic property of objects, that can only change due to some event if this event changes the state of this object.

But the agent can be more precise, because there is more that he knows about being blue. Whether an object is red or blue depends not on the objects ‘entire’ state, but only on its colour, which is a part of this state. The state of the colour of an object can be calculated from the state of the object. To express this formally, the agent needs:
• An appropriate state space, a colour space.

• A selector, which computes a colour in this space, given the state of an object.

Introducing the predicate ‘colour’ that holds for all elements of the colour space, and the selector ‘clr’ from objects to colours, one has:

$$\forall x.\text{colour}(\text{clr}([x\&t]))$$

(4.2)

With the help of this selector the agent can reduce the property ‘bl’ even further. He can define a timeless predicate ‘bb’ over the elements of the colour space that holds for all those colours that are blue. The ‘time-dependent’ predicate ‘blue’ over objects can subsequently be expressed in terms of the eternal predicate ‘bb’ over the colour-space, as follows:

$$\forall x, t. \text{blue}(x\&t) \leftrightarrow \text{bb}(\text{clr}([x\&t]))$$

(4.3)

This equation implies that an object remains ‘blue’ unless its colour changes. Other state changes of the object cannot affect the ‘blueness’ of the object: from \(\text{blue}(a\&t_1)\) and \(\text{clr}([a\&t_1]) = \text{clr}([a\&t_2])\) it follows, with equation 4.3: \(\text{blue}(a\&t_2)\). The agent also knows that the definition of ‘being red’ resembles the definition of ‘being blue’. An object is red if the colour of the object is found in a region of the colour space defined by some predicate \(rr\): 

$$\forall x, t. \text{red}(x\&t) \leftrightarrow \text{rr}(\text{clr}([x\&t]))$$

(4.4)

Also, colours which are blue are not red, so the predicates ‘rr’ and ‘bb’ over colours are such that there is no overlap between the predicates ‘rr’ and ‘bb’ on the colour space, i.e.:

$$\forall s.\text{colour}(s) \land \text{rr}(s) \land \text{bb}(s) \rightarrow \bot$$

From this fact and equations 4.2, 4.3 and 4.4 it subsequently follows that objects cannot be both red and blue at the same time:

$$\forall x, t. \text{red}(x\&t) \land \text{blue}(x\&t) \rightarrow \bot$$

(4.5)

Note that we now have deduced an invariant about objects by combining a timeless fact about the colour space using a reduction of the time-dependent predicates ‘blue’ and ‘red’ to a condition on the state of objects. Binary predicates can also be reduced to conditions on the states of objects. Consider a relation like ‘longer’. Whether or not two objects stand in this relation is as a rule determined by their respective lengths. These lengths can be calculated from the local object state, for instance by a selector: ‘lmg’. Typically, lengths
4.5. **HOW TO REASON**

may be represented as quantities. Then the predicate ‘longer’ can be defined by the following equivalence:

\[ \forall x, y, t_1, t_2. \text{longer}(x & t_1, y & t_2) \leftrightarrow (\text{lng}([x & t_1]) > \text{lng}([y & t_2])) \]  

(4.6)

Note that the formalism does not force the arguments of a binary predicate like ‘longer’ to be simultaneous. This allows facts to be expressed that one cannot handle in most conventional frameworks. The judgement that \( a \) is longer at the moment \( t_1 \) than \( b \) was at moment \( t_2 \), can be expressed as: \( \text{longer}(a & t_1, b & t_2) \). Also, the judgement that \( a \) is longer at \( t_2 \) than it itself was at \( t_1 \) — for instance because it has grown — can be formulated as: \( \text{longer}(a & t_2, a & t_1) \).

4.5.2 **Knowledge about events**

There are a number of things that an agent needs to know about events in order to be able to reason about the dynamics of a certain universe. He has to be able to distinguish the different kinds of events that can take place, and must have some knowledge about the effects of these events, and about the conditions under which they may happen. The agent does not have to know all possible events; it is sufficient to know what kinds of events can change certain attributes or properties of objects.

As an example, we consider some robots that are moving in a given universe, while painting tiles in different colours. In this universe there are at least two kinds of objects: robots and tiles. The predicates ‘robot’ and ‘tile’ distinguish these objects. In order to reason about the movement of a robot, one considers an attribute of the state of a robot: its position. It is assumed that there is a state space whose elements are positions. To express the fact that the position of a robot can be computed from the state of a robot, the selector \( psn \) is introduced, which computes the position of a robot from its state.

Though an agent may not know all the kinds of events that can take place in this universe, the agent may very well know all kinds of events in this universe that can change the position of a robot. For instance, it may be the case that a position can only change if the robot moves. This implies that if the position of a robot before an event differs from its position after the event, this event involves a move of the robot. This can be expressed by the following equation that quantifies over events. (Remember that variables \( e \) always quantify over events.)

\[ \forall r, e. \text{robot}(r) \land (\text{psn}([r \& \alpha(e)]) \neq \text{psn}([r \& \beta(e)])) \rightarrow \text{move}(e, r) \]  

(4.7)

It is also possible to express how the movements of a robot affect its position. For example, the position after a move may always be adjacent
to the position before the move. Assuming a timeless predicate ‘adj’ that expresses adjacency of positions, we get:

\[ \forall e, r. \move(e, r) \rightarrow \adj(psn([r&\alpha(e)]), psn([r&\beta(e)])) \]

(4.8)

If positions can be specified further, for instance by the use of coordinates, more precise information about movement can also be expressed.

To change other attributes, other kinds of events may be needed. For instance, in the universe at hand, the colour of a tile only changes when it is painted by a robot. This can be expressed as follows:

\[ \forall l, e. \tile(l) \land \text{clr}([l&\beta(e)]) \neq \text{clr}([l&\alpha(e)]) \rightarrow \exists r. \robot(r) \land \text{paint}(e, l, r) \]

(4.9)

It is also possible to express preconditions. If a robot can only paint a tile when it is at the same place as the tile before it starts to paint, this can be expressed as follows:

\[ \forall l, e, r. \tile(l) \land \robot(r) \land \text{paint}(e, l, r) \rightarrow \psn([l&\alpha(e)]) = \psn([r&\alpha(e)]) \]

(4.10)

### 4.5.3 Knowledge about time

Time is considered to contain moments and events. As has been discussed, it is assumed that events and moments are linearly ordered, that they are atomic and discrete, that events and moments alternate, and that the number of events between any two moments is finite. The resulting ordering relation ‘≺’ between moments and events is straightforward. To be able to reason about this relation the given assumptions need to be formalised. This can be done in terms of axioms about the special predicate ‘≺’. The axioms that describe this ordering can be divided in two kinds:

- Axioms that hold because moments and events alternate and are ordered linearly.
- Axioms that hold because the number of events between any two moments is finite.

We will never explicitly use the ordering axioms, as reasoning about this ordering is tedious and does not add anything interesting. The axioms are only given for the sake of definiteness:

#### Ordering of moments and events

The variables ‘q’, ‘q1’ and ‘q2’ as used in this axiomatisation can range both over moments and events.
4.5. **HOW TO REASON**

- The ordering is irreflexive:

\[ \forall e. \neg (e \prec e) \]

\[ \forall t. \neg (t \prec t) \]

- It is a total ordering, where moments and events are disjoint categories.

\[ \forall e, t. (e \prec t) \lor (t \prec e) \]

- The ordering is transitive:

\[ \forall q_1, q, q_2. (q_1 \prec q) \land (q \prec q_2) \rightarrow (q_1 \prec q_2) \]

- After every event \( e \) there is a moment \( \alpha(e) \) that is the immediate successor of this event:

\[ \forall e. (e \prec \alpha(e)) \]

\[ \forall e. \exists q. (e \prec q \prec \alpha(e)) \]

- Before every event \( e \) there is a moment \( \beta(e) \) that it its immediate predecessor:

\[ \forall e. (\beta(e) \prec e) \]

\[ \forall e. \exists q. (\beta(e) \prec q \prec e) \]

- Every moment \( t \) has an immediate predecessor \( e \) that is an event:

\[ \forall t. \exists e. (t = \alpha(e)) \]

- Every moment \( t \) has an immediate successor \( e \) that is an event:

\[ \forall t. \exists e. (t = \beta(e)) \]
Finiteness

More interesting is the fact that the ordering \(<\) assumes that there is only a finite number of events between any two moments. It follows that if a property of an object is changed between two moments then there is a first and a last event that has changed this property. These events may coincide.

This can be expressed by axioms that have some similarity with induction axioms. If an object has a property \(P\) at a moment \(t_1\), and does no longer have this property at a later moment \(t_2\) (where \(t_1 < t_2\)) then there must be an event \(e\) (between \(t_1\) and \(t_2\)) that changes this property for the first time. As this is the first event that makes this change, \(P\) still holds at any time \(t\) where \(t_1 < t\) and \(t < e\). Formally, this can be expressed as follows:

\[
\forall x, t_1, t_2. \quad P(x\&t_1) \land (t_1 < t_2) \land \neg P(x\&t_2) \\
\rightarrow \\
(\exists e. \quad (t_1 < e < t_2) \land P(x\&\beta(e)) \land \\
\neg P(x\&\alpha(e)) \land (\forall t. \quad (t_1 < t < e) \rightarrow P(x\&t))) \quad (4.11)
\]

The above axiom also has a dual, which postulates that a property must have held in the past since the last event that has changed it:

\[
\forall x, t_1, t_2. \quad \neg P(x\&t_1) \land (t_1 < t_2) \land P(x\&t_2) \\
\rightarrow \\
(\exists e. \quad (t_1 < e < t_2) \land \neg P(x\&\beta(e)) \land \\
P(x\&\alpha(e)) \land (\forall t. \quad (e < t < t_2) \rightarrow P(x\&t))) \quad (4.12)
\]

4.5.4 Knowledge about particulars

To reason about the history of his universe, an agent needs some knowledge about historic states of affairs: this knowledge consists of a simple collection of facts, some timeless, and some that hold at given moments. For instance:

\[
\text{robot}(r) \\
\text{tile}(l_1) \\
\quad e < t_1 \\
\text{blue}(l_1\&t_1) \\
\text{paint}(e, l_1, r)
\]

and so on.
4.6 Reasoning example

In this section we will present a description of a small universe, that we will gradually extend. This example is intended to achieve two things: on the one hand it is meant to illustrate how an agent can reason about change in the framework that we have put forward; on the other hand we want to convince the reader that the description of a given universe can be extended indefinitely, — which means that the approach lends itself to reason about the dynamics of worlds that are only partially specified. So the method employed is not based on an exhaustive search over events, and can tackle reasoning problems about quite complex situations in a modular fashion.

In the example universe one finds changing objects: some robots are moving about, and sometimes paint the tiles over which they move. Whether they do paint a tile, and in what colour, depends on factors that do not concern us here. The knowledge about this universe is only partial, and the universe is nondeterministic. We will prove a number of facts about the robots in this universe. We start with simple facts, that require only a minimum of knowledge about the universe, and gradually turn to more complicated proofs, in which more detailed knowledge about the robots is added.

4.6.1 Crossing the border

The first proof involves the ‘continuity’ of movement. We want to show that a robot has to cross the border of a certain room if it wants to get from the inside of the room to the outside.

To show this, we have to make a number of assumptions. We will assume that a robot always has a certain position, that this position can only change if the robot moves, and that all movement is between adjacent positions.

To formalise these assumptions, first the kinds of objects that play a role have to be introduced, next the relevant aspects of the state of these objects must be characterised. Also, we have to know which kind of events can change certain aspects of the states of these objects, and how these events affect these states.

We introduce the robots by means of a predicate ‘robot’: \( \text{robot}(x) \) is true if the object \( x \) is a robot. If \( x \) is a robot, and \( t \) is a moment, then the state of the robot at the moment \( t \), i.e. \( [x & t] \), can be arbitrarily complex. In order to prove facts about movement, we are — at present — only interested in one single aspect of the state of the robot: its \textit{position}. So we assume that there is some state space, whose elements are positions. To project the ‘position’ aspect out of the complex robot state, a selector is introduced: ‘\textit{psn}’. Using
a predicate ‘position’ that holds for all states that are positions, we have:

\[ \forall x, t. \; \text{robot}(x) \rightarrow \text{position}(\text{psn}([x\&t])) \]  \hspace{2cm} (4.13)

The reader should note the difference between the predicate ‘robot’ that is a predicate over objects, i.e. historic entities, the selector ‘psn’ that is a function from states to elements of the position space, and the predicate ‘position’ that is a predicate which holds for all elements of the position space.

To prove anything about the dynamics of robot positions, it is essential to know what kind of event can change the position of a robot. We assume that this position can only change if the robot ‘moves’. In other words, if the position of a robot \( r \) before an event \( e \) differs from the position after this event, then this event involves a move of the robot, cf. equation 4.7:

\[ \forall r, e. \; \text{robot}(r) \land (\text{psn}([r\&\beta(e)]) \neq \text{psn}([r\&\alpha(e)])) \rightarrow \text{move}(e, r) \]  \hspace{2cm} (4.14)

How do the movements of a robot affect its position? If we do not specify anything about the direction of movement, all that we know is that if an event involves a robot move, the position of the robot after the event is adjacent to the position of the robot before the event, cf. equation 4.8:

\[ \forall e, r. \; \text{move}(e, r) \rightarrow \text{adj}([r\&\beta(e)], \text{psn}([r\&\alpha(e)])) \]  \hspace{2cm} (4.15)

We want to show that a robot has to cross the border of a certain room if it wants to get from the inside of this room to the outside. To judge whether the robot is inside or outside the room we have to consider its position, and distinguish positions inside the room from positions outside the room. Assuming that there is a timeless predicate ‘room’ over positions that makes this distinction, we can define the predicate ‘inside’ for a given object as:

\[ \forall x, t. \; (\text{inside}(x\&t) \leftrightarrow \text{room}(\text{psn}([x\&t]))) \]  \hspace{2cm} (4.16)

Next we define the border of the room as the set of those positions in the room that are adjacent to positions not in the room. To do so, we define the timeless predicate ‘border’ over positions as follows:

\[ \forall p. \; \text{border}(p) \leftrightarrow \exists q. \; (\text{adj}(p, q) \land \text{room}(p) \land \neg \text{room}(q)) \]  \hspace{2cm} (4.17)

Now we are ready to start the proof. Let’s assume that a robot \( r_1 \) is inside at time \( t_1 \) and outside at a later time \( t_2 \):

\[ \text{robot}(r_1) \land (t_1 < t_2) \land \text{inside}(r_1\&t_1) \land \neg \text{inside}(r_1\&t_2) \]  \hspace{2cm} (4.18)
4.6. REASONING EXAMPLE

Now the proof that the robot has passed the border between \( t_1 \) and \( t_2 \) proceeds as follows. Taking equation 4.11:

\[
\forall x, t_1, t_2. \quad P(x \& t_1) \land (t_1 < t_2) \land \neg P(x \& t_2)
\]

\[
\rightarrow
\]

\[
(\exists e. \quad (t_1 < e < t_2) \land P(x \& \beta(e)) \land 
\neg P(x \& \alpha(e)) \land (\forall t. \quad (t_1 < t < e) \rightarrow P(x \& t)))
\]

and substituting \( P = \text{inside} \), we can conclude:

\[
\exists e_0. \quad (t_1 < e_0 < t_2) \land \text{inside}(r_1 \& \beta(e_0)) \land \neg \text{inside}(r_1 \& \alpha(e_0)) \quad (4.19)
\]

Expanding parts of this formula with the definitions as given by equation 4.16 we get:

\[
\text{room}(\text{psn}([r_1 \& \beta(e_0)])) \land \neg \text{room}(\text{psn}([r_1 \& \alpha(e_0)]))
\]

From this it follows immediately that:

\[
\text{psn}([r_1 \& \beta(e_0)]) \neq \text{psn}([r_1 \& \alpha(e_0)])
\]

Using equation 4.14 we can conclude that the event \( e_0 \) is a move event of the robot:

\[
\text{move}(e_0, r_1)
\]

From this fact we have, by equation 4.15:

\[
\text{adj}(\text{psn}([r_1 \& \beta(e_0)]), \text{psn}([r_1 \& \alpha(e_0)]))
\]

Collecting these results and substituting in the right-hand side of the definition of ‘border’, as given by equation 4.17, yields:

\[
\text{border}(\text{psn}([r_1 \& \beta(e_0)])) \quad (4.20)
\]

This proves that, if the robot \( r_1 \) is in the room at time \( t_1 \) and is outside the room at time \( t_2 \), there is a moment \( \beta(e_0) \) between \( t_1 \) and \( t_2 \) (the moment just before the event of leaving the room) at which the robot is on the border. By abstracting over the assumptions in this example, the result can be generalised to arbitrary robots at arbitrary points in time. Note that the result has been proved without any assumption whatsoever about the sequence of events — involving either this robot, other robots, or other things — between \( t_1 \) and \( t_2 \).
4.6.2 Painting

To prove slightly more interesting results, the example will be extended and some extra complications are introduced: it is assumed that the robots are able to paint the places over which they travel. To describe such a situation a new kind of object is introduced: a ‘tile’. Tiles have a position and a colour.

\[ \forall x, t. \text{tile}(x) \rightarrow \text{position}(\text{psn}([x \& t])) \]  
(4.21)

\[ \forall x, t. \text{tile}(x) \rightarrow \text{colour}(\text{clr}([x \& t])) \]  
(4.22)

Tiles have fixed positions:

\[ \forall t_1, t_2, l. \text{tile}(l) \rightarrow (\text{psn}([l \& t_1]) = \text{psn}([l \& t_2])) \]  
(4.23)

The colour of a tile only changes when painted by a robot, cf. equation 4.9:

\[ \forall l, e. \text{tile}(l) \wedge (\text{clr}([l \& \beta(e)]) \neq \text{clr}([l \& \alpha(e)])) \rightarrow \exists r. \text{robot}(r) \wedge \text{paint}(e, l, r) \]  
(4.24)

A robot can paint a tile only if it is at the right position when it starts to paint, cf. equation 4.10:

\[ \forall l, e, r. \text{tile}(l) \wedge \text{robot}(r) \wedge \text{paint}(e, l, r) \rightarrow (\text{psn}([l \& \beta(e)]) = \text{psn}([r \& \beta(e)])) \]  
(4.25)

The trouble with red paint

If one paints the places over which one moves, this may have the interesting effect that subsequent movements become restricted. To mimic this effect here, it is assumed that tiles are sometimes being painted red — where the predicate ‘red’ will be defined as in equation 4.4 — and that the movement of robots is restricted by the additional constraint that a robot can never be on a red tile:

\[ \forall r, l, t. \text{robot}(r) \wedge \text{tile}(l) \wedge \text{red}(l \& t) \rightarrow (\text{psn}([r \& t]) \neq \text{psn}([l \& t])) \]  
(4.26)

At first sight, such a constraint may seem to pose a problem: how is a robot ever able to paint a certain tile red, if on the one hand, the robot must be at this tile when he starts to paint it, but on the other hand, cannot, at any moment, be found at a red tile? The answer is that it is only specified that the tile in question is red when the painting job is finished, which implies that the robot cannot be at this tile after the painting. Apparently the robot has to paint and move simultaneously whenever it is painting a tile red. As it has never been stated or assumed that a painting event and a moving event
are separate, it is possible that the robot moves and paints in one and the same event. After such an event, the tile is red, and the robot is found on a tile at an adjacent position.

Of course, the constraint as given by equation 4.26 does imply that the robot can no longer move onto tiles once they are red, so it cannot paint them again. So any tile, once painted red, remains red. We will first prove this. Formally, we want to prove:

\[ \forall l, t_1, t_2. \ (\text{tile}(l) \land (t_1 < t_2) \land \text{red}(l \land t_1)) \rightarrow \text{red}(l \land t_2) \]  

(4.27)

It should be noted that the proof that we construct assumes nothing about the number of robots that exist or about the colours that they prefer to paint tiles in. The result depends only on the fact that red tiles become inaccessible to robots, and on the fact (4.24) that all painting is done by robots. To prove equation 4.27, we derive the contradiction from the assumption that a red tile changes colour.

Assume that a tile \( l_2 \) which is red at a moment \( t_5 \) is no longer red at a later moment \( t_6 \):

\[ \text{red}(l_2 \land t_5) \land \neg \text{red}(l_2 \land t_6) \land (t_5 < t_6) \]  

(4.28)

By equation 4.11, there must be an event that changes the colour of this tile for the first time since \( t_5 \):

\[ \exists e_1. \ (t_5 < e_1 < t_6) \land \text{red}(l_2 \land \beta(e_1)) \land \neg \text{red}(l_2 \land \alpha(e_1)) \]

By the same equation:

\[ \forall t. \ (t_5 < t < e_1) \rightarrow \text{red}(l_2 \land t) \]  

(4.29)

Using the definition of redness in terms of colour as given by equation 4.4, this implies that the event \( e_1 \) has changed the colour of the tile \( l_2 \):

\[ \text{clr}([l_2 \land \beta(e_1)]) \neq \text{clr}([l_2 \land \alpha(e_1)]) \]

Therefore, by the fact that colours of tiles only change when robots paint them as given by equation 4.24, there must be a robot that painted the tile \( "l_2" \):

\[ \exists r_x \cdot \text{robot}(r_x) \land \text{paint}(e_1, l_2, r_x) \]

Combining this result with equation 4.25 implies:

\[ \text{psn}([r_x \land \beta(e_1)]) = \text{psn}([l_2 \land \beta(e_1)]) \]
But from the ordering axioms we have $\beta(e_1) \prec e_1$ which, together with equation 4.29, implies:

$$\text{red}(l_2 & \beta(e_1))$$

Substitution of the above results in equation 4.26 leads to contradictory statements about the position of the mysterious robot $r_2$, thereby proving that the temporary assumption 4.28 is false. This concludes the proof of equation 4.27.

### 4.6.3 Painting a room

We now get on with the main gist of the example, and consider the situation where all the tiles in the room are painted. We assume that the robots cannot fly, they are always on the floor (which can be painted), and all positions on the floor correspond to a tile. So if a robot is found at a certain position, there is always a tile corresponding to this position:

$$\forall t, r. \text{robot}(r) \rightarrow \exists l. \text{tile}(l) \land (\text{psn}([l \& t]) = \text{psn}([r \& t]))$$  \hspace{1cm} (4.30)

For the sake of definiteness we further assume that all tiles in the room are blue at a certain time $t_3$ and have been painted red at a later time $t_4$. We thus have the following data.

At $t_3$ all tiles inside the room are blue:

$$\forall l. \text{inside}(l \& t_3) \rightarrow \text{blue}(l \& t_3)$$  \hspace{1cm} (4.31)

At $t_4$ (with $t_2 \prec t_4$) all the tiles inside the room are red:

$$\forall l. \text{inside}(l \& t_4) \rightarrow \text{red}(l \& t_4)$$  \hspace{1cm} (4.32)

To make the case more interesting assume that there is a tile called the doorway:

$$\text{tile}(dw)$$

This is the only tile found at the border of the room:

$$\forall l, t. (\text{tile}(l) \land \text{border}(\text{psn}([l \& t]))) \leftrightarrow (l = dw)$$

In this situation we wish to prove that the last tile painted in the room must be the doorway. In other words: there exist an event in which the doorway is painted by some robot, and this is the last paint event of any tile inside the room.

Informally, we reason as follows: as the room contains at least the tile in the doorway, there is an event $f$ when this tile was painted red. This tile
remains red, which implies that no robot is found at the doorway after $f$. Assume an event $e$ in which a tile $l$ in the room is painted after this event $f$. Then there is a robot $r$ in the room after the event $f$. This robot must always be on a tile, also after moment $t_4$. But as, after $t_4$, all tiles in the room are red, there can no longer be a robot in the room after $t_4$. So the robot $r$ has left the room between $\beta(e)$ and $t_4$. To do so, it must have been at some tile at the border of the room. But the only tile at the border is the doorway. The robot therefore must have been at the doorway after the event $f$, which is impossible, as we have already concluded. Note that we have not assumed that there is only one robot, nor that robots always paint in red.

Formally the proof runs as follows. To be proven:

$$\exists f, r_2. \text{paint}(f, dw, r_2) \land \neg \exists e, r, l. (f \prec e) \land \text{paint}(e, l, r) \land \text{inside}(l \& \beta(e))$$

As the doorway is on the border, it is in the room, and we have, using our data from 4.31 and 4.32 and equation 4.5, which expresses an elementary fact about colours:

$$\neg \text{red}(dw \& t_3)$$

$$\text{red}(dw \& t_4)$$

From this we can conclude, by equation 4.11 under the substitution $P = \neg \text{red}$ that:

$$\exists f. \neg \text{red}(dw \& \beta(f)) \land \text{red}(dw \& \alpha(f))$$

(4.33)

From this fact it follows by equation 4.4 that there is a change of colour at the doorway:

$$\text{clr}([dw \& \beta(f)]) \neq \text{clr}([dw \& \alpha(f)])$$

Together with equation 4.24 this implies that the doorway has been painted by some robot. This proves the left-most part of the demonstrandum:

$$\exists f, r_2. \text{paint}(f, dw, r_2)$$

Given this event $f$, we must now prove the rest of the conjunction, i.e:

$$\neg \exists e, r, l. (f \prec e) \land \text{paint}(e, l, r) \land \text{inside}(l \& \beta(e))$$

Combining equation 4.33 with equation 4.27, the doorway must always remain red after the event $f$:

$$\forall t. (f \prec t) \rightarrow \text{red}(dw \& t)$$
This fact, together with equation 4.26, implies that no robot will ever be found in the doorway after \( f \):

\[
\forall t, r. (f \prec t) \land robot(r) \rightarrow (\text{psn}([r \& t]) \neq \text{psn}([dw \& t]))
\]  

(4.34)

We now assume the opposite of the right part of the demonstrandum; let’s imagine that some tile \( l \) inside the room is painted by some unknown robot \( r \) after the event \( f \):

\[
\exists e, r, l. (f \prec e) \land \text{paint}(e, l, r) \land \text{inside}(l \& \beta(e))
\]

We know about the imaginary painting event \( e \) from equation 4.25 that the unknown robot \( r \) was at the tile \( l \), i.e. inside the room at the moment \( \beta(e) \) just before the event \( e \). From the ordering axioms, we can prove that this moment \( \beta(e) \) lies between the events \( e \) and \( f \):

\[
\text{room}(\text{psn}([r \& \beta(e)])) \land (f \prec \beta(e))
\]

The trouble for the robot \( r \) is that after the moment \( t_4 \) all tiles in the room are red, whereas, by equation 4.30 the robot \( r \) must always be somewhere on some tile, also after the moment \( t_4 \):

\[
\forall t. \exists l. \text{tile}(l) \land (\text{psn}([l \& t]) = \text{psn}([r \& t]))
\]

As robots can never be on a red tile, as expressed by equation 4.26, and as red tiles always remain red, it follows, that there can no longer be any robots in the room after \( t_4 \), when all tiles in the room are red:

\[
\forall t. \text{room}(\text{psn}([r \& t])) \rightarrow (t < t_4)
\]

So we also know that:

\[
\beta(e) \prec t_4
\]

Reasoning in the same way as we did in the first example, we find that the robot \( r \) must have left the room in an event \( g \) before \( t_4 \) but after the moment \( \beta(e) \), when it still was in the room. It can thus be shown that the robot must have been on the border just before it left the room, between the moments \( \beta(e) \) and \( t_4 \):

\[
\exists g. (\beta(e) \prec g \prec t_4) \land \text{border}(\text{psn}([r \& \beta(g)]))
\]

But according to equation 4.30 the robot must have been on some tile at the border. As the only tile at the border is the doorway, the robot must have been at the doorway at the moment \( \beta(g) \):

\[
\text{psn}([dw \& \beta(g)]) = \text{psn}([r \& \beta(g)])
\]

As the moment \( \beta(g) \) is certain to be after \( f \), because we have \( f \prec \beta(e) \) this leads to a contradiction with equation 4.34. This completes the proof of the right part of the demonstrandum. As we have already proved the left part, this completes the proof.
4.6. REASONING EXAMPLE

4.6.4 In the mood

We finally examine the robots in more detail. We assume that the robots can paint in the colours red, white or blue. The robots can also be in different moods. The actions that the robots undertake depend both on their mood and on their surroundings. For instance, a robot paints everything red when it is angry. Also, it only stops being angry when it encounters a white tile. These additional details are described by the following equations.

Angry robots paint in red:

$$\forall r, x, e. \text{robot}(r) \land \text{paint}(e, x, r) \land \text{angry}(r \& \beta(e)) \rightarrow \text{red}(x \& \alpha(e))$$

If an event changes the mood of an angry robot, it has to be at a white tile:

$$\forall e, r. \text{robot}(r) \land \text{angry}(r \& \beta(e)) \land \neg \text{angry}(r \& \alpha(e)) \rightarrow \exists x. \text{tile}(x) \land \text{white}(x \& \beta(e)) \land (\text{psn}([r \& \beta(e)]) = \text{psn}([x \& \beta(e)]))$$

Assume that at time $t_1$ a robot $r_1$ is angry, and that there are no tiles that are already white. We can now prove that, if there are no other robots, our robot will go on to paint in red.

We will only sketch the proof here, as it is somewhat elaborate, and follows a similar pattern as previous proofs. The pattern of the proof is simple. At time $t_1$ there are no white tiles. If the robot $r_1$ is not angry at some later moment, there is a first event $f$ that is responsible for the mood switch. Till this event takes place, the robot will remain angry. (All this follows from equation 4.11.) At the inception of this event $f$ the agent is at a white tile. Call this tile $x$. As this tile was not white at $t_1$, but is white at $\beta(f)$ there must have been event $g$ between $t_1$ and $\beta(f)$ that changed the colour of the tile $x$ to white. This must have been a paint event. As there is only one robot, the (still angry, at least till $f$) robot $r_1$, this robot was involved in the paint event, that took place before $f$. Thus the angry robot has painted the tile white in this event. As angry robots only paint in red, and white things are not red, a paradox ensues.

This example is interesting because it contains a vicious circle: To change the colour in which it paints, the robot needs to get out of the angry mood. But to do so the robot needs a white tile. Unfortunately, it cannot make a white tile by itself, because it is still angry ... The frustrating nature of the problem is well known to anyone that once left his keys in his car. The method we present here has the ability to identify necessary events, and to order them in time, which makes it possible to detect such a vicious circle, and to prove that the situation is hopeless.

Note, that the assumption that there is only one robot, is an essential one. If this is not assumed, one can not show that the robot remains angry,
but one can show other interesting things: for instance, that if there is time $t_1$ at which there is an angry robot that roams a universe which — at that time — has no white tiles, and if there is a later time $t_2$ where this same robot is no longer angry, that there must have been another robot at time $t_1$, and that this other robot was not angry then.

### 4.7 Conclusions and discussion

The examples that we have presented show that the method really works. It is possible to reason about the dynamic objects that inhabit a small artificial universe within the confines of predicate logic, based on a formalisation of the ontology given in section 4.3.

In this ontology, in which events figure as first class citizens, we describe dynamic objects not as indivisible entities, but as changing structures that can be decomposed, enabling us to localise changes within the objects. The resulting formalisation enables us to reason about change and the localised effects of events. As we have seen in the examples, it is possible to prove that certain events have to take place, or that events have to take place in certain order, or that some invariant holds from a given moment on. As the resulting temporal reasoning method is monotonous, and does not rely on exhaustive search, it can even be applied in worlds that are only partially specified. This means that it is possible to reason in an incremental way, coming to more precise conclusions as our knowledge about a situation is refined and extended.

As the method proposed here differs fundamentally from other methods found in the literature, comparison with these methods is rather difficult. Most of the methods found in the literature are simply not suited to deal with the kind of examples that we have given here. For instance, most modal approaches to temporal reasoning seem to limit themselves to the case of propositional logic, or are not yet in a state where they can deal with ‘practical’ examples.

An interesting exception is Lamport’s temporal logic of actions (TLA) [Lamport, 1994]. This is a (modal) logic that can be used to verify both liveness and safety properties for concurrent programs.

Despite the syntactic differences between Lamport’s approach and our own, there are important semantic parallels between the two. TLA has a semantics that is also based on a notion of ‘states’. These states are roughly equivalent to the states of the universe in our approach.

An important difference is that TLA does not support the notion of an ‘object’. Instead of objects, it uses a pool of variables. The state of the ‘universe’ is always given by the current state of all the variables. The meaning of the variables is given through mappings from the TLA-states to values.
Though the values of the variables cannot be decomposed through selectors, one can see from their meaning that they are clearly related to 'attributes' in our own approach. Being a temporal logic, TLA does not allow explicit quantification over moments or events, but the modal operators (that can be given a semantics in terms of such quantification) are quite expressive, and the proof system is very powerful. A decisive difference is that in TLA one cannot quantify over objects (as there are none) and that one cannot quantify over variables either. For this reason it seems TLA can not handle the examples given here.

Other approaches, that do allow quantification over objects, like the situational calculus, are geared towards problem solving, and do not seem to offer the capability to prove invariants at all.

The reasoning method that comes closest to our own seems to be the one of the 'TRAiNS' project, in which Allen's 'interval temporal logic' is used, and where frame axioms are coded through a technique called 'explanation closure', specifying for each property which kind of event can change it. Like we do, Allen also handles moments and events as first class citizens on a par with objects, and uses standard predicate logic to reason about time.

Nevertheless, it is difficult to encode our examples in Allen's framework. For instance, in our first example, we show that a robot that moves from a position 'inside' the room to a position 'outside' the room must have crossed the 'border' of the room. Though Allen's method may be able to handle this example for a given room with a given number of positions and given adjacency relations between these positions, it is not clear to us how the general problem can be dealt with. The problem is that situations are encoded in a different way. The example problems that are handled in the TRAiNS project, where one deals with movements of trains rather than robots, always seem to presuppose a given layout of all available railroad connections.

To be able to make any comparison at all, one has to restrict the general examples that we have handled in our method, and consider specific versions of these examples in which the layout of the room and all tiles and adjacent positions are given. Though we have not actually tried this, it appears to be possible, in such a restricted case, to also prove a version of the second example using Allen's method (i.e. that the last tile painted within a given room must be the tile at the doorway). However, as the room layout has to be given, to be able to conduct such a proof, it is to be expected that the complexity of this proof will depend on the size of the room. This means that the examples will be unwieldy — both to humans and computers — when one has to deal with rooms which contain more than a few tiles.

As is apparent from the above, our method differs substantially from the one used by Allen and his co-workers. There are several minor differences, and there is one very important difference. We will now discuss some of these differences in more detail.
One of the minor differences seems to be responsible for the fact that Allen’s method may have difficulty in dealing with examples that involve a ‘vicious circle’. Where we use equations 4.11 and 4.12 to indicate that there must be a first and last event that is responsible for a given predicate change, Allen uses a somewhat similar axiom that asserts that every property change results from an appropriate event. See, for instance: [Ferguson, 1995] (pp. 31, strong closure on properties). But as this axiom only states that there is an event that changes the property, but does not state that there is a ‘first’ and a ‘last’ event that changes it, this axiom is significantly weaker than the one we use, and cannot be used to exclude an infinite regress. For this reason, it does not allow you to show that certain problems that involve a ‘vicious circle’ are unsolvable.

Other differences stem from the fact that Allen’s logic takes time intervals as basic, where we consider time intervals as derived entities, as stretches of time that are delimited by instantaneous events. To us it seems that all these differences, though significant, are nonetheless minor ones. They can be overcome by straightforward adaptions to either approach. However, there is also one major difference.

This major difference concerns the central issue at stake when reasoning about time: how to understand what change is, and how to determine formally what changes and what doesn’t change as the result of some action. In Allen’s method, as in all reasoning methods that we are acquainted with, it is assumed that change means predicate change: events change the world as they switch propositions on or off. Consequently, there is no underlying changing ‘substance’ from which the (apparent) changes of propositions about the different objects may be derived. This implies that for each and every predicate one has to describe (through some axioms) how it changes as a result of the different events.

By contrast, the formalisation that we propose here, is based on an ontology that does not only recognise objects, events and predicates, but also includes ‘states’ in which an object can be found. In fact, all changes take place because objects move through their state spaces. The effects of different kinds of events are given through changes that they effect in the states of objects. The changing predicates and properties, on the other hand, are defined in terms of conditions on the states of objects, i.e. they correspond to regions in state spaces. The way in which the truth values of propositions that involve these predicates changes due to different kinds of events can subsequently be derived.

Formally, this difference is reflected in the fact that we use explicit references to states of objects, and use ‘selectors’ to decompose these states. This allows certain facts to be expressed that are outside the scope of other methods. For instance, one can express in our formalisation that an object
$x$ has not changed due to an event $\epsilon$:

$$[x & \beta(\epsilon)] = [x & \alpha(\epsilon)]$$

From this it can subsequently be concluded that all properties that are defined in terms of selectors over this state – the intrinsic properties of the object – are also not affected by $\epsilon$. Such rather powerful arguments, that acknowledge the decomposable nature of objects, subsequently allow us to localise the effects of different events. By contrast, temporal reasoning methods that do not use states treat objects syntactically, as mere arguments of propositions. They do not provide us with the formal tools to express what objects really are, i.e. things (that have parts) that can change. Accordingly, they are not able to express whether a certain event does or does not affect a given object.

For this reason these methods only work if the events that can occur are somehow delimited. Typically, one takes the perspective of a specific agent, and distinguishes 'planned' events (which are supposed to occur) and 'external' or 'spontaneous' events, that are to be avoided. At worst, the external and spontaneous events are simply not considered. At best, one assumes that there are 'event closure' axioms, that strongly limit the events that can occur. These totally exclude or strongly delimit the 'external' and 'spontaneous' events. On the basis of event closure assumptions it is subsequently possible to reason about the things that may happen, applying what is essentially a form of exhaustive reasoning.

By contrast, we do not use exhaustive methods. The fact that 'external' or 'spontaneous' events can happen does not constitute a big problem, because our method allows the effects of events to be localised. Accordingly, our method is able to function in situations that are strongly underspecified, as we have seen in the various examples: the layout of rooms and doorways, the number of robots, tiles or other things, were to a very large extent undetermined.

Considering how our non-exhaustive method works, it is quite clear that it has certain strong limitations. The method simply collects all the evidence about the history of the universe that is available. Based on this evidence one can then prove facts, provided these facts hold in all possible histories of the universe that are in accordance with the given evidence. This means that the method can only prove that which is necessary, like the fact that an invariant holds, or that a certain events must take place. It is not possible to prove that a certain event can happen. Given the fact that the method can be formalised in a monotonic framework, this fact is not really surprising.\(^6\)

\(^6\)There may be an interesting way out of this, however. In physical systems, there is no independent information about future events and states, and the question whether or not a certain event can happen only depends on the current state of objects that involve
CHAPTER 4. MODELLING DYNAMIC OBJECTS

Another question regards the generality of the proposed method. Though it is very difficult to give a formal answer to this question, there are clear indications that the range of applicability of the method is rather large. This follows from a simple observation.

The painting robots that we have described are in fact rather universal machines. They have an internal state (their mood), various output symbols (the different colours) and are responsive to input symbols at different locations. In fact, they are similar to Turing machines. If we assume that there is precisely one robot, that it paints in certain colours, dependent upon its current mood and the last colour it has moved over, this robot is a perfect instance of the classical case. Our examples show, therefore, that the method can be used to prove properties about Turing machines. In fact, in the last example we have proved, under certain assumptions about the program and the tape, that a particular class of Turing machines does never reach a certain internal state.

It is to be expected that the method can be used to reason about small computer programs. It not only enables one to prove properties of Turing machines, but it also supports a view in which objects are seen as entities that undergo local state changes, and it seems quite easy and natural to specify object-oriented programs in the formalism. There is a deep reason for this. As we have seen, the proposed encoding of dynamic worlds in predicate logic rests on the idea that properties of objects are reduced to conditions on the states of these objects. But this reduction of properties to conditions on states is also essential to the functioning of computer programs. Here, the state of various objects is always explicitly represented in terms of data structures, and all the relevant properties of the objects — those boolean switches that determine what happens next — are expressed as conditions on the (state of the) objects. The processor evaluates these conditions in order to choose how to proceed. So the idea proposed in this chapter is nothing but a formal analysis of a state-based view on change, a view that is quite close to the intuitive view that programmers may take, in particular when working within the object-oriented paradigm. Indeed, the so-called frame problem that always seems to arise when one tries to describe a changing universe in a formal logic, does not seem to have a counterpart in (conventional) programming language semantics, where objects are represented by changing structures. The method presented here seems to be able to exploit this fact.

---

If all events only have local and instantaneous preconditions, one may define a notion of ‘possibility’ that reflects ‘physical possibility’, and in which the question what ‘can’ happen depends only on the actual physical state. Thus we are able to prove that a certain type of event ‘can’ happen — in this restricted physical sense — by proving that the actual state meets the preconditions of this type of event. Of course, this physical notion of possibility is a monotonous one. If the robot can paint the tile, but actually does not paint the tile, it later still is the case that it could have painted the tile.
Chapter 5

Time, types and memory

It's a poor sort of memory that only works backwards.

The White Queen

In this chapter we will apply the method developed in the previous chapter to the kind of knowledge states that we investigated in chapter 3. The idea is to model the knowledge state of an agent that is able to deal with time, who can observe events and states of affairs, and can reason about the past, the present and the future.

To this end, we will first revisit the knowledge state $\Gamma_A$ of the agent $A$ that we investigated in chapter 3. We will adapt and extend this knowledge state in such a way that the agent can use it to reason about the changing universe with the painting robots that we encountered in the previous chapter. The formal changes that we have to make in the basic parts of the knowledge state in order to achieve this, are rather limited. Still, the knowledge state will become more complicated, as knowledge about events and the structure of time has to be incorporated. Therefore, some additions are necessary.

5.1 Adaptation of the knowledge state

To see which changes are necessary and how they work, we first examine how the types and functions in the small example knowledge state $\Gamma_A$ given in chapter 3 can be reinterpreted in terms of the ontology and notation of the previous chapter. Fortunately, large parts of the knowledge state do not need to be reinterpreted at all. In particular, the ‘logical’ and the ‘mathematical’ ‘layers’ in the knowledge state are not time dependent in any way, and do not need to be changed or reinterpreted. But the dynamic interpretation of several types that occur in the conceptual layer needs to be examined.
5.1.1  Reinterpreting the conceptual layer

We first recall the conceptual layer of the context $\Gamma_A$ as presented in subsection 3.3.3.

Things

The conceptual part of the context $\Gamma_A$ starts with the introduction of things. Things, and in particular robots and tiles, are declared using the following types:

\[ \text{thing} : * \]
\[ \text{robot} < \text{thing} : * \]
\[ \text{tile} < \text{thing} : * \]

We want to interpret these types within the ontology presented in chapter 4. Clearly, the type ‘thing’ and also the types ‘robot’ and ‘tile’ must be somehow related to objects. We have seen in chapter 4 that there are at least three views on objects. Correspondingly, these types might be considered as referring to historical objects, to their states, or to time slices. Since the ‘meaning’ of the types ‘robot’ and ‘tile’ resides in an agents ability to recognise robots and tiles, we have to reconsider what such recognition means in a more dynamical setting, where things can change. What we see when we recognise a robot at a certain moment in time is clearly not the object as a historical entity; our observation is limited to the ‘now’ and can only concern the ‘combination’ of the historical object with the present moment. So what we recognise is a time slice, in the sense of chapter 4; we recognise the robot in its present circumstances.

This is quite fortunate, because, as we have seen, the most fundamental level at which objects can be considered in the dynamic ontology is the level that does not abstract over anything, the level of time slices. Accordingly, we interpret the type ‘thing’ as referring to time slices of objects, the type ‘robot’ as referring to time slices of robots and the type ‘tile’ as referring to time slices of tiles. The decision that we have taken here is a rather crucial one, which has many consequences for the construction of knowledge states. Time slices are rather special in comparison to the more traditional mathematical values that we will encounter elsewhere; there are things one can do with time slices, that we cannot do with other values. To get a clear separation between those types that are to be interpreted as time slices and those that correspond to the more traditional mathematical values, we will make a minor change in the introduction of the types that correspond to time slices. We introduce an extra sort $*_t$ that contains the types that correspond to time slices. This might seem to be an unnecessary complication,
5.1. ADAPTATION OF THE KNOWLEDGE STATE

but the advantages will become apparent in section 5.2.2. Accordingly, the introductions of ‘things’, ‘robots’ and ‘tiles’ will now be as follows:

\[
\text{thing} : *_t \\
\text{robot} < \text{thing} : *_t \\
\text{tile} < \text{thing} : *_t
\]

A consequence of using two sorts is that we need two versions of Leibniz-equality; the familiar equality ‘eq’ defined for types in ‘*’, and its analogon for types in ‘*\_t’:

\[
eq_{@} := (\lambda t.*_t. (\lambda x,y.t.(\Pi g:(\Pi z:*_t). (g \ x) \iff (g \ y)))) : (\Pi t.*_t.(\Pi z:*_t))
\]

We now proceed with the investigation and reinterpretation of the context \( \Gamma_A \).

Attributes and state spaces

The context \( \Gamma_A \) continues with the declaration of the different kinds of attributes:

\[
\text{colour} : * \\
\text{position} : * \\
\text{length} : * \\
\text{mood} : * \\
\text{intensity} : *
\]

Types like ‘colour’,‘intensity’,‘position’,‘length,’ and ‘mood’ are best reinterpreted as referring to state spaces. As state spaces are static, this poses no problems. It is natural to interpret the \textit{inhabitants} of such types as the (intensional counterparts of the) elements of state spaces. Some state spaces are declared to be quantitative:

\[
q : (\text{quant length}) \\
p : (\text{quant intensity})
\]

This fact, i.e. that they have inhabitants that can be compared, does not change in a dynamic setting, and therefore poses no interpretation problems. Functions like ‘r’,‘g’,‘b’, that allow us to decompose attributes in terms of more primitive attributes, pose no problem either. They can be interpreted as selectors, i.e. as functions between state spaces. In this example these functions go from type colour – the colour space – to type intensity, i.e. they correspond to selectors from the colour space to the intensity space.
CHAPTER 5. TIME, TYPES AND MEMORY

\[ r : \text{colour} \Rightarrow \text{intensity} \]
\[ g : \text{colour} \Rightarrow \text{intensity} \]
\[ b : \text{colour} \Rightarrow \text{intensity} \]

Some delicate functions

Next, the context \( \Gamma_A \) continues with the declaration of a few functions that
relate ‘things’ to their ‘colour’, ‘position’, or ‘length’:

\[ \text{pos} : \text{thing} \Rightarrow \text{position} \]
\[ \text{col} : \text{thing} \Rightarrow \text{colour} \]
\[ \text{len} : \text{robot} \Rightarrow \text{length} \]

In the formalism that we developed in chapter 4, we do have functions like
‘col’ that go from the type ‘thing’ – now interpreted as referring to time slices
of things – to ‘colour’ that refers to a colour space, but there are no symbols
for them. The reason is that these functions can only be formed by componing
other, more basic functions. In the last chapter we have encountered the
function ‘\( \lambda x. [x] \)’ that goes from time slices to states, and the function ‘\( \text{clr} \)’
that goes from states to the elements of the colour space. By composing these
two functions, we can construct the function ‘\( \text{col} \)’, as \( \text{col} = (\lambda x. \text{clr}([x])) \). The
resulting function goes from time slices to colours, and this is the function
that we need to interpret the function ‘\( \text{col} \)’. In the same way, we can construct
a function that can be used to interpret ‘\( \text{pos} \)’, where \( \text{pos} = (\lambda x. \text{psn}([x])) \). For
other functions, like ‘\( \text{len} \)’ similar constructions are possible.

Predicates on attributes

The context \( \Gamma_A \) in section 3.3.3 also contains definitions of some predicates
over attributes, like the definition of redness, blueness, whiteness etc. of
different colours:

\[ rr := (\lambda x : \text{colour.} \]
\[ (\text{gt intensity} p (r x)(g x)) \land \]
\[ (\text{gt intensity} p (r x)(b x))) : (\text{colour} \rightarrow *) \]

\[ gg := (\lambda x : \text{colour.} \]
\[ (\text{gt intensity} p (g x)(r x)) \land \]
\[ (\text{gt intensity} p (g x)(b x))) : (\text{colour} \rightarrow *) \]
5.2. EXTENSIONS OF THE KNOWLEDGE STATE

\[ bb := (\lambda x : colour. (gt \text{ intensity p}(b \times(r \times)) \land (gt \text{ intensity p}(b \times(g \times)) : (colour \rightarrow *)) \]

\[ ww := (\lambda x : colour. (eq \text{ intensity p}(g \times(r \times)) \land (eq \text{ intensity p}(r \times(b \times))) : (colour \rightarrow *)) \]

These definitions only contain terms that we already know how to interpret and go through without any problem.

**Predicates on things**

The context also contains predicates on things like ‘longer’, ‘red’ and ‘grn’:

\[ longer := (\lambda x, y : thing.(gt \text{ length q}(len \times)(len \times)) : (\Pi x, yr \text{ robot}.*)) \]
\[ red := (\lambda x : thing.(rr \text{ (col} \times)) : (\Pi x:thing.*) \]
\[ grn := (\lambda x:thing.(gg \text{ (col} \times)) : (\Pi x:thing.*) \]
\[ blue := (\lambda x : thing.(bb \text{ (col} \times)) : (\Pi x:thing.*) \]
\[ white := (\lambda x : thing.(ww \text{ (col} \times)) : (\Pi x:thing.*) \]

These predicates have definitions, so we can understand how they have to be interpreted from the way they are defined, i.e. from an understanding of the interpretation of the definiens. The key to understanding the interpretation of these predicates is a proper understanding of the functions like ‘col’, ‘pos’ and ‘len’ that we already discussed.

This ends the re-interpretation of the conceptual layer of the knowledge state represented by the context \( \Gamma_A \).

### 5.2 Extensions of the knowledge state

The next question is how the factual knowledge of an agent is formalised in type theory. Let us first consider the factual knowledge we see in section 3.3.4. The question is how to express, in the dynamical setting of chapter 4, such facts as: “There is only one red robot”, or “Marvin is longer than c3po”.

It is important to note, first, that these simple facts become rather incomplete in a dynamic setting, for what does it now mean that there is only one red robot? It can mean that there is only one red robot now, or that there is always one red robot, whose identity might be subject to change. Finally, it can mean that there is precisely one red robot and that this is
always the same robot. The introduction of time clearly necessitates a more precise interpretation of a statement. Even for a simple assertion like: “Marvin paints a tile”, one may argue that there is now an ambiguity as to the time of the tile-painting.

Another factor that complicates the formulation of facts in a dynamic setting, is that we have to consider new kinds of facts. For instance, there are now also facts about the ordering of different moments and events in time, and facts about the effects of different events. We must be able express facts like: “Event $e_1$ takes place after moment $t_2$”, and “A robot whose position has changed, has moved”.

To be able to deal with all these matters, we have to incorporate concepts that allow us to express knowledge about time, about the relations between the time slices of objects at different moments, and about the effects and conditions of various events.

5.2.1 Time

In order to be able to represent statements about time, we have to construct types that harbour moments and events as presented in the previous chapter. It may be fair to say that we construct a basic ‘temporal’ level between the ‘mathematical’ and the ‘conceptual’ level. A way to introduce moments and events is to declare the following types:

\[
\text{time} : * \\
\text{qt} : (\text{quant time}) \\
\text{event} < \text{time} : * \\
\text{moment} < \text{time} : * \\
\text{\textless} := (gt, qt, time) : \text{time} \rightarrow \text{time} \rightarrow *
\]

In the sequel, we will use infix notation for ‘\textless’.

As in section 4.4.1 we introduce two functions from events to moments; one function ‘$\beta$’ that, given an event, yields the moment just before the event, and another function ‘$\alpha$’, that, given an event, yields the moment just after the event:

\[
\beta : \text{event} \rightarrow \text{moment}, \\
\alpha : \text{event} \rightarrow \text{moment}
\]
5.2. EXTENSIONS OF THE KNOWLEDGE STATE

We also need to declare the relevant axioms about the structure of time, as given in section 4.5.3. We postpone this for the time being, and first concentrate on the treatment of time slices. The remaining questions about the structure of time are taken up again in section 5.2.4.

5.2.2 Time slices

It is essential to the method presented in chapter 4 that one is able to refer to the time slice of a certain object just before or just after a certain event. We therefore also need this ability in the new context \( \Gamma_A \) that we are developing. How can we extend this knowledge state in such a way that it allows us to construct different time slices of the same object, or simultaneous time slices of different objects?

One possibility is to declare special types that stand for historical objects – as opposed to time slices of objects – and introduce a function that acts like the function ‘&’ that we use in chapter 4. Another, more attractive option is to consider historical objects as equivalence classes of time slices. This is the option that we will choose. It has the advantage that it avoids the introduction of historical objects as an extra primitive notion.

Constructing time slices

Since we choose not to introduce historical objects, we are barred from introducing a function ‘&’ from historical objects to time slices. We use a function from time slices and moments to time slices. We write this function as ‘\( \downarrow \)’. The function ‘\( \downarrow \)’ allows us to ‘jump’ from a time slice (its first argument) to a corresponding time slice of the same object at a given moment (its second argument). In the terminology of chapter 4, this function has the property that:

\[
\forall t_1, t_2.((x \& t_1) \downarrow t_2) = (x \& t_2)
\]

and is clearly well-defined\(^1\) in the model given in the previous chapter.

But when we want to introduce such a function in a knowledge state \( \Gamma_A \) we notice a small problem. Trying to determine its type, we see that we need several such functions; for instance, in the robot universe that has ‘things’, ‘robots’ and ‘tiles’ we need the following functions:

\[
\downarrow_{\text{thing}} : \text{thing} \rightarrow (\text{moment} \rightarrow \text{thing})
\]

\(^1\)Time slices are pairs, the function \( \downarrow \) takes the object from the first argument (by taking the first component of the time slice that is a pair) and forms a pair using its second argument.
\( \downarrow_{\text{tile}} : \text{tile} \rightarrow (\text{moment} \rightarrow \text{tile}) \)
\( \downarrow_{\text{robot}} : \text{robot} \rightarrow (\text{moment} \rightarrow \text{robot}) \)

This 'problem' can be solved quite easily by the introduction of a polymorphic function \('\downarrow'\) that accepts as an extra argument the types of time slices. In the given example these types are: 'thing', 'robot' and 'tile'. As we have chosen to introduce the types of time slices as inhabitants of a special second sort \('*_{t}'\), we can now indeed restrict the quantification of types in a polymorphic function to those types that contain time slices. Accordingly, the polymorphic function \('\downarrow'\) can be introduced as:

\[ \downarrow : (\Pi t:*_{t}.(\Pi x:t.\Pi m:\text{moment}.) t) \]

or, alternatively, in clause form:

\[ [t:*_{t}, x:t, m:\text{moment}] \Rightarrow (\downarrow t x m): t \]

In the sequel, when referring to time slices, we will often allow ourselves to suppress the type argument and write \('\downarrow'\) instead of e.g. \('\downarrow_{\text{robot}}'\). Also, we will use infix notation for the function \('\downarrow'\), even in the type-theoretical formalisation.

**Relating different time slices**

By itself, having the function \('\downarrow'\) is not enough. To be able to discuss the historical relationship between time slices – to distinguish those time slices that make up the same historical object – one needs to introduce an equivalence relation between time slices. This relation also needs to be polymorphic. We declare it as follows:

\[ [t:*_{t}, x:t, y:t] \Rightarrow (\text{same } t x y) : * \]

Every time slice is given at a certain moment, and we want to be able to refer to this particular moment, for instance to express that time slices are simultaneous. To do so we introduce a (polymorphic) function \('\text{now}'\) that yields the moment at which the time slice is actual:

\[ [t:*_{t}, x:t] \Rightarrow (\text{now } t x) : \text{moment} \]

There are many ways to axiomatise the properties of the equivalence relation 'same' and the function 'now'. The following axioms are rather straightforward, but the reader is warned that they may not be the most successful
ones to use in automatic theorem provers:  

\[
[t : *_{t}, m : \text{moment}, x : t, y : t, q : (e_{q_{2}} t (x \downarrow m) y)] \\
\Rightarrow (\text{hist } t m x y q) : (\text{same } t x y)
\]  

(5.1)

\[
[t : *_{t}, m : \text{moment}, x : t, y : t, q : (\text{same } t x y)] \\
\Rightarrow (\text{alt } t m x y q) : (e_{q_{2}} t (x \downarrow m) (y \downarrow m))
\]

(5.2)

\[
[t : *_{t}, m_{1} : \text{moment}, m_{2} : \text{moment}, x : t, y : t, q : (e_{q_{2}} t (x \downarrow m_{1}) (y \downarrow m_{2})] \\
\Rightarrow (\text{simul } t m_{1} m_{2} x y q) : (e_{q} \text{moment } m_{1} m_{2})
\]

(5.3)

\[
[t : *_{t}, r : t] \Rightarrow (\text{sel } t r) : (e_{q_{2}} t r (r \downarrow \text{(now } t r)))
\]

(5.4)

Note again that the use of the sort ‘*_{t}’ allows us to specify that the given functions and relations apply only to time slices.

5.2.3 Expressing facts

Armed with the functions: ‘\(\downarrow\)’, ‘now’ and the equivalence relation ‘same’, we are now ready to formalise various facts. Let us start with the fact from chapter 3 that there is only one red robot. In its most stringent reading, where this robot is always the ‘same’ robot, this fact can now be expressed as follows:

\[
[r_{1} : \text{robot}, r_{2} : \text{robot}, q : (\text{red } r_{1}), q_{2} : (\text{red } r_{2})] \\
\Rightarrow (\text{uniq } r_{1} r_{2} q q_{2}) : (\text{same robot } r_{1} r_{2}),
\]

It might be the case that the colour of robots never changes; this can now be expressed as follows;

\[
[r_{1} : \text{robot}, r_{2} : \text{robot}, q : (\text{same robot } r_{1} r_{2})] \\
\Rightarrow (\text{mov } r_{1} r_{2} q) : (\text{eq colour } (\text{col } r_{1}) (\text{col } r_{2}))
\]

How can we express that a robot must have moved if it has changed its position in some event? First, we have to declare a predicate ‘move’ over

\footnote{For instance, it seems to be more efficient to code the equivalence relation ‘same’ for elements of some type \(T\) through the introduction of a subtype for \(T\) for each historic individual. However, the details of this idea need still to be worked out.}
events and robots.

\[\text{move : event } \rightarrow \text{robot } \rightarrow *\]

The rule corresponding to equation 4.14 of the previous chapter can now be expressed through the following clause:

\[\left[ r : \text{robot}, e : \text{event}, q : \neg(eq\ position(pos(r \downarrow (\beta e)))(pos(r \downarrow (\alpha e)))) \right] \Rightarrow (ax_{34} r e q) : (move e r)\]

The translation of other equations given in chapter 4 follows the same pattern. In principle, we can now translate all these rules, and extend the knowledge state \(\Gamma_A\) accordingly. For instance, after we have defined adjacency as a predicate between positions:

\[\text{adj : position } \rightarrow \text{position } \rightarrow *\]

we can encode the fact that all movements are between adjacent positions in the following clause.

\[\left[ e : \text{event}, r : \text{robot}, q : (move e r) \right] \Rightarrow (ax_{35} e r q) : (\text{adj}(pos(r \downarrow (\beta e)))(pos(r \downarrow (\alpha e))))\]

All these translations, and the corresponding extensions of the knowledge state \(\Gamma_A\), are relatively straightforward. But before we can reason about time in the extended knowledge state \(\Gamma_A\), we first have to show how one can translate the remaining knowledge about the structure of time as given in section 4.5.3 into type theory.

5.2.4 More about time

We need to formalise the information that we have about the structure of time given in section 4.5.3. The translation of the axioms of part one is quite simple, and some examples should suffice:

\[\left[ q : \text{time}, p : (q \prec q) \right] \Rightarrow (irref q p) : \bot\]

\[\left[ e : \text{event} \right] \Rightarrow (inception e) : ((\beta e) \prec e)\]

\[\left[ e : \text{event} \right] \Rightarrow (conclusion e) : (e \prec (\alpha e))\]

... The case for the two axioms of part two of the same section is somewhat
more complicated, as these axioms also involve the function ‘&’. We therefore show in some detail how they are to be translated. As both axioms are very similar, it is sufficient to translate the first one, given by equation 4.11. This axiom states that if a property of an object has changed between two moments in time, that there is always a most early event that has changed it. The axiom reads as follows:

\[
\forall x, t_1, t_2. P(x \& t_1) \land (t_1 < t_2) \land \neg P(x \& t_2) \\
\rightarrow \\
(\exists e. (t_1 < e < t_2) \land P(x \& \beta(e)) \land \\
\neg P(x \& \alpha(e)) \land (\forall t. (t_1 < t < e) \rightarrow P(x \& t)))
\]  

(5.5)

As the function ‘&’ is not available in the TT translation we have to use the function ‘|’. This poses no problems. A possible complicating factor is the cluttering of conclusions in the consequence of the axiom. This is caused by the scope of the existential quantifier for the postulated event ‘e’ in question.

Using a Skolem function ‘early_{\theta_0}’ to express the existence of this event, the axiom can be split into a number of separate clauses. We present these clauses here for the curious:

\[d : *x, r : d, t_1 : \text{moment}, t_2 : \text{moment},
P : (\Pi x : d, *), q : (P(r \downarrow d t_1)),
z : (t_1 < t_2), q2 : \neg(P(r \downarrow d t_2))
\Rightarrow
(early_{\theta_0} d \ r \ t_1 \ t_2 \ p \ q \ z \ q_2) : \text{event}
\]

\[d : *x, r : d, t_1 : \text{moment}, t_2 : \text{moment},
P : (\Pi x : d, *), q : (P(r \downarrow d t_1)),
z : (t_1 < t_2), q2 : \neg(P(r \downarrow d t_2))
\Rightarrow
(early_{\theta_1} d \ r \ t_1 \ t_2 \ p \ q \ z \ q_2) : (t_1 < (early_{\theta_0} d \ r \ t_1 \ t_2 \ p \ q \ z \ q_2))
\]

\[d : *x, r : d, t_1 : \text{moment}, t_2 : \text{moment},
P : (\Pi x : d, *), q : (P(r \downarrow d t_1)),
z : (t_1 < t_2), q2 : \neg(P(r \downarrow d t_2))
\Rightarrow
(early_{\theta_2} d \ r \ t_1 \ t_2 \ p \ q \ z \ q_2) : ((early_{\theta_0} d \ r \ t_1 \ t_2 \ p \ q \ z \ q_2) \prec t_2)
\]

\[d : *x, r : d, t_1 : \text{moment}, t_2 : \text{moment},
P : (\Pi x : d, *), q : (P(r \downarrow d t_1)),
z : (t_1 < t_2), q2 : \neg(P(r \downarrow d t_2))
\]
\[ (\text{early} y_3 \; d \; r \; t_1 \; t_2 \; p \; q \; z \; q_2) : (P(r \downarrow_d (\beta(\text{early} y_0 \; d \; r \; t_1 \; t_2 \; p \; q \; z \; q_2)))) \]

\[ [d : \{ r \}; r : d, t_1 : \text{moment}, t_2 : \text{moment};
P : (\Pi_{d \times r}), q : (P (r \downarrow_d t_1)),
z : (t_1 \prec t_2), q_2 : \neg(P(r \downarrow_d t_2)) \]
\[ \Rightarrow (\text{early} y_4 \; d \; r \; t_1 \; t_2 \; p \; q \; z \; q_2) : (\neg(P(r \downarrow_d (\alpha(\text{early} y_0 \; d \; r \; t_1 \; t_2 \; p \; q \; z \; q_2)))) \]

\[ [d : \{ r \}; r : d, t_1 : \text{moment}, t_2 : \text{moment};
P : (\Pi_{d \times r}), q : (P (r \downarrow_d t_1)),
z : (t_1 \prec t_2), q_2 : \neg(P(r \downarrow_d t_2)),
t : \text{moment}, g : (t_1 \prec t),
h : (t \prec (\text{early} y_0 \; d \; r \; t_1 \; t_2 \; p \; q \; z \; q_2)) \]
\[ \Rightarrow (\text{early} y_5 \; d \; r \; t_1 \; t_2 \; p \; q \; z \; q_2 \; t \; g \; h) : (P (r \downarrow_d t)) \]

This concludes the description of the adaptations that we need to make to the knowledge state \( \Gamma_A \) as given in chapter 3 in order to be able to reason about time.

## 5.3 Observation and reasoning

If we want to model the cognitive ability of the agent, we have to understand how the knowledge state of the agent is extended with observations. For the case of a 'static' external world, this has been discussed in section 3.7.2. The dynamics of this process is entirely straightforward. Given a successful observation of an observable type \( T \), the agent just extends his context with a fresh inhabitant of this type. But how will observation work in the dynamic case?

To answer this question we consider an agent \( A \) with a knowledge state \( \Gamma_A \). As in chapter 3 the agent can make observations; the observational abilities of the agent were specified in chapter 3 through a set \( \mathcal{O} \) of interpretable primitives:

\[ \mathcal{O} = \{ \text{thing}, \; \text{robot}, \; \text{tile}, \; (\text{gt intensity} \; p), \; (\text{eq intensity}), \]
\[ (\text{gt length} \; q), \; (\text{eq length}), \; \text{loc}, \; \text{col}, \; r, \; g, \; b \} \]

To make sufficiently interesting observations in the changing robot world, the set of interpretable primitives for the agent \( A \) has to be somewhat larger. We must extend it in such a way as to enable the agent to observe whether
5.3.  OBSERVATION AND REASONING

robots are at a certain tile, whether they are inside or outside the room, etc. There are various ways to go about this. The most straightforward way is to endow the agent with the ability to compare positions. This may be realised in several ways; the simplest is to make a position decomposable in terms of quantifiable ‘x’ and ‘y’ coordinates, much in the same way as the ‘colour’ can be composed into quantifiable spectral intensities. The reader will have no difficulty to see how this may be done. In this way, the agent will have far-reaching observational abilities, enabling him, for instance to see whether a robot is inside the room (see equation 4.16), what colour it has, on what colour of tile it is standing, etc.

Besides, an agent A that really lives ‘in time’ must have some extra observational abilities, to deal with time itself. In particular, he needs some way to order his observations in time. Though the question how the agent may be able to do so in ‘realistic’ terms is rather interesting, we have neither time nor space to go into it here. Simplifying matters considerably, we therefore assume that the agent can, whenever he makes an observation, identify the moment at which this observation is made, and is able to compare and identify such moments. The reader may imagine that the agent is able to consult a clock.

5.3.1 Observation example

We now consider how the agent A may fare in a specific case. Let \( t_1 \) be a past moment. Assume that A has, at the moment \( t_1 \), made the observation that there is a red robot inside the room. Formally, this means that the agent has observed (an inhabitant of) the type:

\[
(\Sigma r : robot. (\Sigma q : (red \ r). (\Sigma p : (inside \ r). (eq \ moment (now \ robot \ r) \ t_1))))
\]

Using Procedure 2, discussed in section 3.7.2 the agent has succeeded in recording this observation and within his context \( \Gamma_A \) has declared an inhabitant (say \( x_1 \)) of this type.

Let \( t_2 \) be a later moment, i.e. \( t_1 < t_2 \). Assume that at \( t_2 \) the agent A again makes the observation that there is a red robot, but this time, it is not inside the room. The observed type is:

\[
(\Sigma r : robot. (\Sigma q : (red \ r). (\Sigma p : (eq \ moment (now \ robot \ r) \ t_2))))
\]

The agent records this new observation using the fresh variable ‘\( x_2 \)’. What can the agent now do with the result of these observations? He can draw conclusions about them, using his reasoning faculties. We consider an example.
5.3.2 Reasoning example

Assume that the context $\Gamma_A$ contains the objects $t_1$, $t_2$, $x_1$ and $x_2$ related to the observations discussed above. In order to reach conclusions about the dynamic behaviour of objects using his rather fragmentary observations, the agent $A$ has to do two things:

- Establish the relations between the various isolated observations, in particular, identify and relate the time slices of objects in the different observations.

- Draw conclusions about the dynamic behaviour of the underlying historic objects.

As we know, if the agent $A$ reasons, he extends his knowledge state with a definition; this has been described in Procedure 1 (section 3.7.1). He may extend his context with the following definitions, that express facts which can be derived:

$$r_1 := \pi_1(x_1) : \text{robot}$$

$$r_2 := \pi_1(x_2) : \text{robot}$$

$$w_1 := \pi_1(\pi_2(x_1)) : (\text{red } r_1)$$

$$w_2 := \pi_1(\pi_2(x_2)) : (\text{red } r_2)$$

$$u_1 := \pi_1(\pi_2(\pi_2(x_1))) : (\text{inside } r_1)$$

$$u_2 := \pi_1(\pi_2(\pi_2(x_2))) : \neg (\text{inside } r_2)$$

$$v_1 := \pi_2(\pi_2(\pi_2(x_1))) : (eq \text{ moment (now robot } r_1) t_1)$$

$$v_2 := \pi_2(\pi_2(\pi_2(x_2))) : (eq \text{ moment (now robot } r_2) t_2)$$

How can the agent $A$ conclude that the time slices $r_1$ and $r_2$ are time slices of the same robot? He knows this, because he knows that there is only one red robot. The formal justification of this fact is constructed using the first fact of section 5.2.3. In the context $\Gamma_A$ extended with the above definitions one has:
5.3. OBSERVATION AND REASONING

\[ \Gamma_A \vdash (\text{uniq } r_1 r_2 w_1 w_2) : (\text{same robot } r_1 r_2) \]

To record this in the context, \( \Gamma_A \) can again be extended with a definition, e.g.:

\[ \text{id} := (\text{uniq } r_1 r_2 w_1 w_2) : (\text{same robot } r_1 r_2) \]

On the basis of this fact one can now relate the various time slices. Using the definition ‘id’ that shows that \( r_1 \) and \( r_2 \) are time slices of the same robot, we can show, using equation 5.2, that \( (r_2 \downarrow t_2) \) is \((r_1 \downarrow t_2)\).

\[ (\text{alt robot } t_2 r_2 r_1 \text{id}) : (\text{eq robot } (r_2 \downarrow t_2) (r_1 \downarrow t_2)) \]

Also, from

\[ v_2 : (\text{eq moment (now robot } r_2) t_2) \]

and

\[ v_1 : (\text{eq moment (now robot } r_1) t_1) \]

we can now show, with equation 5.4, that \( r_2 \) is \((r_2 \downarrow t_2)\) and \( r_1 \) is \((r_1 \downarrow t_1)\).

With some equality reasoning it follows that \( r_2 \) is \((r_1 \downarrow t_2)\) and, also, from the definitions \( u_1 \) and \( u_2 \) that

\[ (\text{inside}(r_1 \downarrow t_1)) \]

and that

\[ \neg(\text{inside}(r_1 \downarrow t_2)) \]

Reasoning in this way, the agent has now extended his knowledge state with a number of conclusions that are remarkably similar to the basic data about a given robot with which we started our first dynamical reasoning example in chapter 4. According to the agent, there is a robot ‘\( r_1 \)’ that is inside a room at a time ‘\( t_1 \)’ and outside at a time ‘\( t_2 \)’. If our translation into TT works, the agent must be able to reason about time within his knowledge state, and reach a similar conclusion as the one we reached in chapter 4. This is indeed the case. To show this with some rigour, we have constructed a TT equivalent of the first example proof of chapter 4. Formally, the argument has been laid down in a ‘book’, that has been checked by a checker that is a part of the DenK system, and that has been reproduced in appendix A.

Summarising, we see that it is possible to extend the subjective knowledge states that we modelled in chapter 3 to deal with a universe that contains changing objects.
5.4 Past, present and future

Considering the situation of the agent $A$ with his extended knowledge state $\Gamma_A$, we see that this agent can now reason about the changing external reality, a reality that he can also — at least to some extent — observe. How does the knowledge state of such an agent develop over time? It is clear that the knowledge state $\Gamma_A$ now contains information about different moments in time. The formal structure of the knowledge state itself makes no principled difference between present, past and future. Of course, the different facts and events can be placed in time, and can be ordered, but the concept of a 'now' seems to be missing. This may, at first sight, seem somewhat strange. It should be noted, however, that not all knowledge states $\Gamma_A$ that are well-formed in the logical sense, can also arise practically.

To see what kind of knowledge states will arise in a real situation it is instructive to imagine an agent that is collecting information about his environment. We assume, for the sake of argument, that the agent is not able to communicate or to form new concepts, and is not able to make inductive generalisations. The agent only reasons with the knowledge that he has, which is continuously complemented with the results of new observations.

When we try to understand what such an agent might be said to know, we see that his knowledge can be divided into observational facts and inborn facts and rules. If the agent is not able to form new concepts or to make inductive generalizations, he is not able to come up with new rules. The only reason why the implicit knowledge of the agent increases, is his observational ability. But there is a limit to the things that the agent can observe. He may have observed facts in the past, and he may have recorded these in his knowledge state, and he may be busy recording incoming observational facts, but he can no longer make any (direct) observations about the past, and he certainly cannot make any observations of the future. The agent can only observe the facts of the present.

So the difference between past, present and future is a subjective distinction which results from the agent's personal observational position, and the fact that his knowledge only stems from the past. To see the whole picture it is important to remember that the knowledge state of the agent also develops in time, mainly because the agent records his observations. At each moment in time the agent has a knowledge state that contains observationally founded facts. But this collection of observational facts stretches from the past to a specific point in time, where it is suddenly cut off, at the moment of the last observation. This cut-off marks a temporal observational horizon, and it is this horizon that the agent may well call the 'now'.
5.4.1 About the future

The possibilities for the agent to ‘know’ the future are rather limited. The agent can infer some information about the future, combining information of present and past with rules that connect different moments in time. But, under normal circumstances, the knowledge of an agent leaves the future largely undetermined. The future of the agent is still largely ‘open’\(^3\).

However, this should not be taken to mean that the agent cannot reason about the future in a useful way. Even though the future is hard to predict, the agent can use his reasoning ability to ponder the consequences and preconditions of future actions. In this thesis, we do not consider actions, but it is interesting to see what happens if the agent sets himself a certain goal that he would like to see realised in the future.

Consider an agent that is in a room full of tiles, where all the tiles are blue. The time is \(t_3\). Let us assume that the agent wants all the tiles in the room to be red at a future time \(t_4\). To consider the consequences of his desire, the agent only has to entertain the assumption that his wish will be fulfilled. Based on this assumption, the agent may now draw various conclusions. For instance, he can conclude that all the tiles in the room have to be painted. But the agent can draw more intricate conclusions. The situation that we sketch here is entirely similar to the one that we encountered in section 4.6.3. Depending on the layout of adjacent locations in the room, the agent can draw conclusions about the order in which various tiles in the room have to be painted. For example, assuming there is only one doorway, a formal proof that the tile found at the doorway has to be painted last has already been given.

Of course, as we have seen several times in chapter 4, it can also happen that the assumption that a certain wish is realised leads to an inconsistency in the knowledge state. This means that the agent can conclude that he has to renounce his goal\(^4\). The goal is impossible to achieve. The fact that the agent can detect this unpleasant fact, without having to resort to some futile exhaustive search, is clearly very useful.

5.5 Conclusions

What we have shown so far is how to construct a knowledge state of an agent that can observe an external reality. We have seen how the agent’s knowledge

---

\(^3\)Interestingly, in the picture that is sketched here, the ‘openness’ of the future is largely an epistemic phenomenon, which is a direct consequence of the agent’s limits of observation.

\(^4\)Assuming that the knowledge of the agent is fully reliable.
can be meaningfully related to an external reality, and how this knowledge may grow, and be continually updated in the course of time. In this way we have indicated how one can model the knowledge state of an agent that is able to integrate reason and observation in a dynamic environment. The agent can observe events and states of affairs and reason about the past, the present and the future. Among other things, this enables the agent to conclude that certain goals can never be achieved, that certain events can preclude others from happening, or that specific events have to take place before certain goals can ever be achieved.

5.5.1 Going on

Thus far, we have only been considering an agent that contemplates the external world on his own. In the following two chapters, we would like to go beyond this, and try to understand how such an agent might communicate with other agents. As we discussed in the first chapter, we will not investigate this question starting from first principles, but assume that there is some language that the agents can use and which functions as a vehicle of communication.

To bridge the gap between type theory and linguistic theory, we will first, in the next chapter, relate type-theoretical knowledge states to the semantic representations of an established linguistic theory, DRT.

Finally, in chapter 7, we will investigate how agents with type-theoretical knowledge states may use language to exchange information, and may develop a common view of their external reality.
Chapter 6

Type theory and DRT

In this chapter we relate type-theoretical knowledge states to the semantic representations of an existing theory of natural language interpretation, Discourse Representation Theory (DRT). To this end, we construct a mapping that translates a Discourse Representation Structure (DRS) into a type. Under this mapping, the model-theoretic truth conditions of DRT translate in a natural way into satisfiability conditions for a type with respect to a given context. In this way one can also produce a semantics that mimics the classical semantics of DRT; to do so, one considers the satisfiability of a type with respect to a complete context, representing the knowledge of an omniscient observer Ω. This indicates that DRSs and their (classical) semantics can be seen as a limit case of a more general theory. In this general theory, utterances relate to types that are well-formed on a given context. These types can be given a semantics based on satisfiability with respect to the context of a given agent, a context that represents this agent's knowledge about the world.

6.1 Introduction

So far, we have been considering an agent that contemplates the external world on his own. Now we consider what happens when an agent tries to communicate with others. We will investigate this against the background of natural language communication. To do so, we will first, in this chapter, relate TT knowledge states to an existing theory of natural language interpretation, Discourse Representation Theory (DRT) [Kamp, 1981], [Kamp & Reyle, 1993].

DRT is based on a view of meaning which has been expressed by Kamp [Kamp, 1981, p. 277] as follows:

Two conceptions of meaning have dominated formal semantics of natural language. The first of these sees meaning principally as
that which determines conditions of truth. This notion, whose advocates are found mostly among philosophers and logicians, has inspired the disciplines of truth-theoretic and model-theoretic semantics. According to the second conception meaning is, first and foremost, that which a language user grasps when he understands the words he hears or reads. This second conception is implicit in many studies by computer scientists (especially those involved with artificial intelligence), psychologists and linguists - studies which have been concerned to articulate the structure of the representations which speakers construct in response to verbal inputs. It appears that these two conceptions, and with them the theoretical concerns that derive from them, have remained largely separated for a considerable period of time. This separation has become an obstacle to the development of semantic theory, impeding progress on either side of the line of division it has created. The theory presented here is an attempt to remove this obstacle.

DRT implements this view on meaning by specifying on the one hand how representations, the DRSs, can be constructed that capture the growing knowledge of a reader, and how these are updated with every new sentence that is interpreted, while on the other hand giving a truth-conditional meaning definition for DRSs. This definition relates them to a model of an external world.

DRSs have an intuitive graphical notation that is easy to understand, and a wide variety of natural language constructs has been given an adequate treatment in DRT, including many forms of anaphora and presuppositions. [vdSandt, 1989], [vdSandt, 1992], [Krahmer, 1995], [Piwek, 1998].

In DRT, the information state of an agent that reads a text is represented by a DRS. The interpretation of discourse is an incremental process. The agent starts with an empty DRS, that is updated with each sentence that he reads. In this way, the agent constructs an 'overall' DRS that represents the discourse read so far. One way to describe this process is to assume that the agent constructs a small DRS for every new sentence that he reads, which is subsequently 'merged' into the overall DRS.

Our goal in this chapter is to show that it is possible to map this process to a similar process on type-theoretical knowledge states. An agent's knowledge is given by a TT context, which is updated with new declarations with each new sentence that he reads. To this end, we will show that it is possible to translate DRSs into types in type theory. We present an algorithm: 'T' that translates DRSs to types in TT. The availability of this translation has a number of pleasant consequences, for instance:

- By composing an algorithm that translates natural language to DRSs
with the translation `T' from DRSs to types, we are able to convert natural language utterances to types in TT.

- As we can map DRSs on types in TT, we can study the formal criteria that play a role in anaphora resolution and presupposition projection in DRT in the context of type theory, enabling us to convert them to similar, but possibly more general criteria that operate in the TT framework. [Piwek, 1998], [Krause, 1995], [Krahmer & Piwek, 1999], [Piwek & Krahmer, 2000].

- On the basis of this translation it is possible to understand how the semantics of DRT can be related to a semantics for TT knowledge states that is not based on truth but on subjective judgement.

- On a more speculative note, the translation enables us to relate discourse semantics to the TT knowledge states of agents that are involved in this discourse, paving the way for an understanding of discourse in terms of the exchange of information between the participants.

The rest of this chapter is organised as follows. First, a formal description of DRSs and their semantics is given. Then two example DRSs are presented, and it is discussed how they may be translated into TT types. Next, we present the algorithm that translates DRSs into TT. Finally we discuss how the semantics of DRSs is related to a semantics in terms of the satisfiability of types.

## 6.2 Discourse Representation Structures

In this section we give a formal description of the syntax and the semantics of DRSs. This presentation largely follows that of [Krahmer, 1995], with the main difference that disjunction and negation have been eliminated, using a non-satisfiable condition represented as `⊥', like in for instance, [Zeevat, 1991].

### 6.2.1 Syntax

A DRS consists of two parts:

- a sequence of discourse referents.
- a sequence of conditions.

Assuming a set $V$ of variables and a set $C$ of constants, the sequence of discourse referents are represented by a list $[r_1\ldots r_n]$ of variables.
A sequence of conditions is represented by a list: \([\phi_1 \ldots \phi_n]\). An empty list is written as \([\ ]\).

A DRS is written as:

\[ [r_1 \ldots r_n, \phi_1 \ldots \phi_m] \]

where \(r_1\ldots r_n\) are discourse referents, \(\phi_1 \ldots \phi_m\) are conditions, and \(m \geq 0\) and \(n \geq 0\). The syntax of a DRS is defined recursively as follows:

- if \(t \in C\) or \(t \in V\) then \(t\) is a term.
- if \(R\) is an \(n\)-ary predicate and \(t_1 \ldots t_n\) are terms, then \(R(t_1, \ldots t_n)\) is a condition.
- if \(D_a\) and \(D_b\) are DRS’s then \((D_a \Rightarrow D_b)\) is a condition.
- if \(t_1\) and \(t_2\) are terms then \((t_1 = t_2)\) is a condition.
- \(\bot\) is a condition.
- if \(r_1\ldots r_n\) are variables, and \(\phi_1 \ldots \phi_n\) are conditions, then \([r_1 \ldots r_n, \phi_1 \ldots \phi_n]\) is a DRS.

### 6.2.2 Truth conditions

A DRS is interpreted with respect to a first-order model \(M\). The model \(M\) is a pair: \(\langle D, I \rangle\) where \(D\), the domain, is a nonempty set, and \(I\), the interpretation, is a function that assigns to each constant in \(C\) an element of \(D\), and to each predicate \(R\) of arity \(n\) an \(n\)-ary relation over elements of \(D\).

In order to determine the truth of a DRS, its discourse referents have to be related to the entities in the model. This relation is established through an assignment.

**Definition 1 (Assignment)** An assignment is a partial function from \(V\) to \(D\).

**Definition 2 (Empty assignment)** The empty assignment \(\Lambda\) is the assignment with an empty domain.

Any (partial) assignment can be extended:

**Definition 3 (Extending an assignment)** Let \(\{r_1 \ldots r_n\} \subseteq V\). An assignment \(g\) extends an assignment \(f\), written as: \(f \leq \{r_1 \ldots r_n\} g\), iff \(\text{domain}(g) = \text{domain}(f) \cup \{r_1 \ldots r_n\}\) and for all variables \(x: x \in \text{domain}(f)\) implies \(f(x) = g(x)\). Thus, an extension \(g\) of an assignment \(f\) is exactly like \(f\), but in addition also assigns values to \(r_1 \ldots r_n\).
6.2. DISCOURSE REPRESENTATION STRUCTURES

To determine that a DRS \([r_1, \ldots, r_n, \phi_1, \ldots, \phi_m] \) is true, one has to find an assignment for the variables \(r_i\) such that all the conditions \(\phi_i\) hold in the model.

When interpreting a discourse, an agent successively constructs DRSs, where a later DRS can contain occurrences of discourse referents that are introduced in earlier DRSs. In accordance with this incremental view, the meaning of a DRS \([r_1, \ldots, r_n, \phi_1, \ldots, \phi_m] \) lies in the way that the DRS allows the agent to extend a given assignment \(g\) — an assignment that he has so far constructed for interpreting the previous DRSs — to a new assignment \(h\), where \(g \subseteq \{r_1, \ldots, r_n\} \ h\).

Formally, the meaning of a DRS \(D\), written as \([D]\), is given by a set of pairs of assignments \((g, h)\) such that \(h\) is an extension of the assignment \(g\), involving the discourse referents in the DRS \(D\). More precisely, for a DRS : \([x_1, \ldots, x_n, \phi_1, \ldots, \phi_m]\), it will be the case that:

\[
\forall g, h. (g, h) \in [[x_1, \ldots, x_n, \phi_1, \ldots, \phi_m]]\Rightarrow g \subseteq \{x_1, \ldots, x_n\} \ h
\]

The meaning of DRSs is defined recursively. We first define the meaning of terms.

**Definition 4 (Meaning of terms)** The meaning of terms is only defined given an assignment. Let \(g\) be an assignment, then \([t]_g = g(t)\) if \(t \in V\) and \([t]_g = I(t)\) if \(t \in C\). If \(g(x)\) is undefined — because \(x\) is outside the domain of \(g\) — then \([x]_g\) is undefined as well.

**Definition 5 (Meaning of a DRS)** For general DRSs the meaning is defined through the following recursive equations. (Note that the meaning of a DRS is defined as a set of pairs of assignments, and the meaning of a condition as the set of all assignments under which this condition holds.)

- \([R(d_1, \ldots, d_n)] = \{g \mid \exists [d_i]_g \in I(R)\}\)
- \([d_1 = d_2] = \{g \mid \exists [d_1]_g, [d_2]_g \in I\}
- \([1] = \{\}\)
- \([D_a \Rightarrow D_b] = \{g \mid \forall h. (g, h) \in [[D_a]] \Rightarrow \exists k. (h, k) \in [[D_b]]\}\)
- \([[x_1, \ldots, x_n \mid \phi_1, \ldots, \phi_m]] = \{g, h\mid (g \subseteq \{x_1, \ldots, x_n\} \ h) \wedge h \in [[\phi_1] \cap \ldots \cap \phi_m]\}\)

Given this definition, we are ready to relate a DRS to a model. A DRS is supported by a given model, given an initial assignment \(g\), if the assignment \(g\) can be extended in such a way that all the conditions in the DRS are true.

**Definition 6 (Supported)** A DRS \(D\) is supported in a model \(W\), with respect to an initial assignment \(g\), if \(\exists h. (g, h) \in [[D]]_W\).
A DRS is true, if there is an assignment \( h \) that extends the initial empty assignment \( \Lambda \), i.e. the assignment that the agent has at the beginning of the discourse, so to speak.

**Definition 7 (Truth)** A DRS \( D \) is true in a model \( W \) iff it is supported in \( W \) with respect to the initial assignment \( \Lambda \).

### 6.2.3 Examples

To get some feeling for the syntax and semantics of DRSs, we will present two examples.

#### Example 1

This example relates to the discourse formed by the following two sentences:

1. *A man whistles.*
2. *A dog follows him.*

A straightforward attempt to translate these sentences into predicate logic runs into quantifier binding problems. DRT presents us with a systematic translation of such sentences into DRSs. The translation deals with the scoping of quantifiers in such a way that it does not suffer from this binding problem, and preserves the intended meaning.

As DRSs can become quite large, it is customary to represent DRSs in a graphical format, that makes for somewhat easier reading. Such DRSs are boxes that are split in two by a horizontal line. In the top box of a DRS we find the discourse referents; in the case at hand, \( \tau_1 \) and \( \tau_2 \). In the second box, we find the conditions. Using this format, the simple DRS that results from an understanding of the given discourse is shown in Figure 6.1.

As has been said, DRSs are interpreted with respect to a first-order model \( M \). The DRS is true if one can find entities in \( M \) which can be assigned to the discourse referents in the top box, such that the conditions in the other boxes are satisfied. Consider the DRS at hand. The variables in the top box
are $r_1$ and $r_2$. There is a verifying assignment for this DRS in the model $M$ if there are entities in the model that can be assigned to $r_1$ and $r_2$ such that the entity assigned to $r_1$ is indeed a man that whistles, and the entity that is assigned to $r_2$ a dog that follows this man.

**Example 2**

The second example is the representation that results from an understanding of one of the famous ‘donkey sentences’ discussed by Geach [Geach, 1962]:

**3 If a farmer owns a donkey, he beats it.**

Again, straightforward translation into predicate logic does not reproduce the intended meaning. This time the variables in the consequent are not bound by the existential quantifiers introduced in the antecedent. In DRT, the sentence in question is translated into the DRS given in Figure 6.2. Note that in this DRS the top box is empty, and that there is only one condition. The condition is a nested one, that contains two sub-DRSs.

As the top box of the DRS is empty, there are no variables here to assign; we simply have to determine whether all the conditions of the DRS are true. There is in fact one single nested condition, which consists of two sub-DRSs. A nested condition is true iff, for any assignment which satisfies the antecedent of the condition, there exists an extension of this assignment for the consequent DRS that also satisfies the consequent. In this case the top box of the consequent is empty, hence the DRS is true if any assignment to $r_1$ and $r_2$ that gives us a farmer and a donkey such that the farmer owns a donkey, is such that the farmer beats the donkey. Any model $M$ in which
this is the case supports the given DRS. As the reader can check for himself, the intended meaning of sentence 3 is thus reproduced.

6.3 Translation of examples

In this section, we investigate how DRSs can be translated into types in TT. We will first consider translations of the two examples.

6.3.1 Example 1

Intuitively, the first DRS, corresponding to the whistling man followed by a dog, rather resembles the following extending segment:

\[ [m : \text{man}, d : \text{dog}, q : (\text{whistles} m), p : (\text{follows} d m)] \]

Let us examine what happens if we translate it in this way. We know that a DRSs is true iff there exist entities in the model that can be assigned to the variables in its top box such that all its conditions are true. Under what conditions can we assert that an extending segment is true? This is not possible; given its subjective foundation, type theory does not provide us with a direct handle on truth. All that it has to offer is a subjective judgement, made by some agent \( A \), the judge.

Any judgement that the judge pronounces about a given segment will necessarily be rooted in his own knowledge. To pronounce a judgement, the judge will therefore have to relate this segment to his own context \( \Gamma_A \), and examine whether the segment has a realisation on his context. We have seen in chapter 3 that an agent 'believes' what an extending segment of his knowledge state expresses, if this extending segment has a realisation on his knowledge state. But there is a complication here. The judge can only judge the given segment if his context \( \Gamma_A \) already contains declarations of variables for the types 'dog' and 'man', and the predicates 'whistles' and 'follows' that occurred in the original DRS. This is typical: apart from the sorts, type theory knows no constants. If one tries to translate languages that use constants into type theory, one has to assume that these constants can be made to correspond to variables that are bound in the context in which the type is to be judged. For the present case, this means that the constants 'man', 'dog', 'whistles' and 'follows' that occur in the DRS must correspond to types (i.e. concepts) that are declared in the context \( \Gamma_A \). The agent \( A \) can establish this correspondence if the vocabulary of a language that he knows (e.g. English) allows him to relate\(^1\) concepts that are declared in his knowledge state to public constants like 'man', 'dog' and 'follows' that

\(^1\)In section 7.2.2 a more explicit account of this process is given.
6.3. **TRANSLATION OF EXAMPLES**

occur in the DRS. So this ‘complication’ is a natural consequence of the private nature of the knowledge state of the agent. An agent has to know the vocabulary of some language to be able to ‘understand’ utterances that use constants from this language.

If we assume that the context of the judge does indeed include the required types, and that the judge speaks a language that allows him to make the proper identifications, the judge will judge that the segment:

\[
[m : \text{man}, d : \text{dog}, q : (\text{whistles } m), p : (\text{follows } d \text{ } m)]
\]

(6.1)
is ‘true’ if he can construct a realisation for this segment. This implies: that there are terms \(x_1, x_2, x_3, x_4\) such that:

\[
\Gamma_A \vdash x_1 : \text{man} \\
\Gamma_A \vdash x_2 : \text{dog} \\
\Gamma_A \vdash x_3 : (\text{whistles } x_1) \\
\Gamma_A \vdash x_4 : (\text{follows } x_2, x_1)
\]

This is the case if the judge knows about a man and a dog, such that the man whistles and the dog is following the man. Thus, the question whether there exists an assignment for the discourse referents of the DRS in the model such that all conditions are satisfied, is now replaced by the question whether the judge knows an actual man and dog, for which he also knows that the conditions are met. If the judge does know this, he will be able to find a realisation for the extension of his context\(^2\).

We conclude that it makes sense to translate the given DRS into the proposed extending segment. But DRSs are tree-like recursive structures, and segments are not. To be able to generalise the proposed translation, DRSs have to be translated to *types*, and not to segments. This is easy, because, as explained in section 2.5.1, one can form types that correspond to extending segments by using \(\Sigma\)-types, i.e. types whose inhabitants are *pairs* of objects. Remember that an inhabitant of a type \(\Sigma x : A.B\) is a pair \((a, b)\) such that \(a : A\) and \(b : B[a/x]\), and that projection operators \(\pi_1\) and \(\pi_2\) extract the left and right components of a pair.

A simple calculation shows that the type that corresponds to the segment 6.1 is the following:

\[
(\Sigma m : \text{man}.(\Sigma d : \text{dog}.(\Sigma q : (\text{whistles } m). (\text{follows } d \text{ } m))))
\]

If the segment 6.1 has a realisation, this type is inhabited.

\(^2\)A far more interesting situation arises if the judge does not know this. We will discuss this in section 6.5.2
6.3.2 Example 2

The second example concerns the DRS of Figure 6.2, which contains two sub-DRSs. Applying what we learned from the first example, we see that the antecedent DRS can be translated to the following type:

\[
(\Sigma f : \text{farmer.}(\Sigma d : \text{donkey.}(\text{owns } f d)))
\]

To see how we can translate the DRS in its entirety it is instructive to assume that there exists an inhabitant \(x\) for the type of the antecedent:

\[
x : (\Sigma f : \text{farmer.}(\Sigma d : \text{donkey.}(\text{owns } f d)))
\]

This assumed inhabitant \(x\) allows us to identify the farmer and the donkey that it introduces: \(\pi_1(x) : \text{farmer}\) and \(\pi_1(\pi_2(x)) : \text{donkey}\). Given this assumption, the intended consequence of the DRS can be expressed as:

\[
(\text{beats } \pi_1(x) \pi_1(\pi_2(x)))
\]

Retracting the assumption we translate the DRS of Figure 6.2 by the type of a function that proves the consequence if the antecedent is assumed. This type is the following:

\[
(\Pi x : (\Sigma f : \text{farmer.}(\Sigma d : \text{donkey.}(\text{owns } f d))).(\text{beats } \pi_1(x) \pi_1(\pi_2(x))))
\]

Though this type looks complicated, because of the projection operators \(\pi_1\) and \(\pi_2\), there is nothing deep in the use of the projection operators, they are only used to make the right connections within the function type. They are needed because the variable ‘\(x\)’ that is introduced in the \(\Pi\)-type is the inhabitant of a complex \(\Sigma\)-type containing farmer, donkey and the owning, all in one. The projection operators help to ‘extract’ the right entities from the inhabitant of this complex type. The use of these operators amounts to a form of ‘plumbing’. Unfortunately, this kind of ‘plumbing’ cannot be avoided when we are using \(\Pi\)-types, and the fact that one has to keep track of variables to ensure that the right connections are made, is the sole factor that complicates the otherwise rather dull and quite straightforward translation from DRT into type theory.

6.4 Formalising the translation

What we have learned from these examples is that DRSs may be translated into types, types that are well-formed on a context \(\Gamma\) on which they may be judged. We also have gained some understanding of the basic principles behind such a translation.
6.4. FORMALISING THE TRANSLATION

Before we can formalise this, and proceed with the general case, we first have to remove a small obstacle. For something seems to be wrong: if we compare the example DRSs with their translations, we see that an important part of the DRSs has not been translated, namely the discourse referents.

The reason for this apparent omission is that type theory is more expressive, due to its types. In TT one can use types like 'man' and 'dog' to declare variables that act like discourse referents. Therefore we could dispense with separate declarations for the discourse referents. But in order to arrive at a systematic translation, it is easier to give up this expressive advantage, and use predicates like 'man' and 'dog' instead. Of course, in TT every variable has a type, and if 'man and 'dog' are defined as predicates, they must be predicates over something, i.e. over some basic type. To cater for this need, we define the basic type 'entity'. So, for the sake of generality, DRSs are translated to slightly more complicated \(\Sigma\)-types then in the examples. For instance, a type like: \((\Sigma x : \text{man.}(\ldots))\) will now be replaced by \((\Sigma x : \text{entity.}(\Sigma q : (\text{man} x).\ldots)))\).

Two further points merit our attention. The first is the presence of constants in the DRSs, which poses a small problem in the translation. We have already encountered this problem when we tried to translate the examples. In type theory there are no constants, apart from the sorts; the type theoretical 'equivalent' of a constant is a variable that is bound in a given context.

To be able to translate the constants that can occur in a DRS, we assume that the 'target' context contains declarations that introduces concepts that correspond to the various constants that may occur in the DRS, and that the agent knows a language that allows him to make the proper identifications. Assuming all this, we simply translate constants literally, as we did in the two examples.

A second point is that a complex condition of the form: \(D_A \Rightarrow D_B\) will translate to a \(\Pi\)-type. As we have seen when translating the DRS for the donkey-sentence, it can be rather complicated to access the right entities that are introduced in a \(\Pi\)-type. Therefore, a systematic translation algorithm has to keep track of information that is needed to carry out this plumbing successfully, i.e. it has to associate discourse referents in DRSs with variables in TT or with projections (using projection operators) of variables.

6.4.1 The translation function \(T\)

We present the translation algorithm in the form of a recursive function \(T\) that computes a type in TT, given a DRS. It is defined by means of a number of recursive rules that rewrite DRSs, conditions, and terms. For clarity we use subscripts \(D, C, t\) that distinguish the rewrite rules: rewrite rules '\(T_D\)' rewrite DRSs, rules '\(T_C\)' rewrite conditions, and rules '\(T_t\)' rewrite terms. All these rules need a list of substitutions \(S\), that they take as an extra ar-
argument, to keep track of the ‘plumbing’ information during the translation. For instance the rule \( \mathbf{T}_D(D, S) \) computes the translation of an (arbitrarily nested) DRS \( D \), given the list of substitutions \( S \). When translation of a DRS is started, on the outermost DRS, the list of substitutions \( S \) is empty. So the translation of a DRS \( D \), i.e. \( \mathbf{T}(D) \) is given by the expression \( \mathbf{T}_D(D, []) \).

To construct the list of substitutions, the algorithm uses an auxiliary function ‘append’ that concatenates two lists. This function works as usual, for instance: \( \text{append}([a, b, c], [d, e]) = [a, b, c, d, e] \).

The function \( \mathbf{T} \) is defined as follows:

- One starts by rewriting a DRS, given an empty list of substitutions:
  \[ \mathbf{T}(D) = \mathbf{T}_D(D, []) \]

- A DRS consisting of a lonely discourse referent translates into the type \( \textit{entity} \):
  \[ \mathbf{T}_D([r_i], S)) = \textit{entity} \]

- A DRS consisting of a lonely condition translates into a single type, which is the translation of that condition:
  \[ \mathbf{T}_D([\phi_i], S) = \mathbf{T}_C(\phi_i, S) \]

- A DRS starting with a list of discourse referents is translated into a \( \Sigma \)-type, in which a fresh variable of type \( \textit{entity} \) is introduced. Simultaneously, the list of substitutions is extended with a pair indicating which discourse referent corresponds to the introduced variable:
  \[ \mathbf{T}_D([r_i \ldots r_n, \phi_1 \ldots \phi_n], S) = (\Sigma x : \textit{entity} . \mathbf{T}_D([r_{i+1} \ldots r_n, \phi_1 \ldots \phi_n], Z)) \]
  where \( x \) fresh and \( Z = \text{append}([x/r_i], S) \).

- A DRS starting with a condition is also translated into a \( \Sigma \)-type. A fresh variable, that functions as a proof object for the type that results from a translation of the given condition, is introduced in this \( \Sigma \)-type. As DRSs do not keep track of proofs, the list of substitutions is not extended:
  \[ \mathbf{T}_D([\phi_1 \ldots \phi_n], S) = (\Sigma x : \mathbf{T}_C(\phi_i, S) . \mathbf{T}_D([\phi_{i+1} \ldots \phi_n], S)) \]
  where \( x \) fresh.
6.4. FORMALISING THE TRANSLATION

- Conditions involving predicates translate under the assumption that predicates can be translated literally:
  \[
  T_C(R(d_1...d_n), S) = (RT_t(d_1, S), ..., T_t(d_n, S))
  \]

- Discourse referents that occur in conditions are translated to the variables or expressions that they have been associated with on the list of substitutions:
  \[
  T_t(r_j, [E/r_1..., F/r_j...G/r_n]) = F
  \]

- Constants are translated literally:
  \[
  T_t(a, S) = a
  \]

- Conditions involving equality are translated in terms of Leibniz-equality.
  \[
  T_C(t_i = t_j, S) = (eq \text{entity } T_t(t_i, S) T_t(t_j, S))
  \]

- The unsatisfiable condition \( \bot \) is translated to the type \( \bot \), i.e. the type of the paradox, as given in section 2.4.2:
  \[
  T_C(\bot, S) = \bot
  \]

- Complex conditions are translated into \( \Pi \)-types, introducing a fresh variable \( x_0 \):
  \[
  T_C(D_a \Rightarrow D_b, S) = (\Pi x_0 : T_D(D_a, S), T_D(D_b, Z))
  \]

  This last rule contains a list of substitutions \( Z \), which is an extension of the list \( S \).

  Let the DRS \( D_a \) be: \([r_1...r_n, \phi_1...\phi_m]\), that is, it contains the discourse referents \( \{r_1, ..., r_n\} \) in its top box. Then \( Z \) is append(X,S) where X is:
  \[
  X = [\pi_1(x_0)/r_1,
  \pi_1(\pi_2(x_0))/r_2,
  \pi_1(\pi_2(\pi_2(x_0)))/r_3,
  ...
  \pi_1(\underbrace{\pi_2(\pi_2(\ldots(\pi_2(x_0)))}_{(n-1)\times})/r_n]
  \]

  Note that this translation is rather straightforward, except for the fact that we need the list of substitutions. This is just a technicality, providing a ‘temporary memory’ that allows the function \( T \) to store the correspondence between a variable that acts as a discourse referent in a DRS, and a sequence of \( \pi \) projections that picks out that part of an inhabitant of a \( \Sigma \)-type that corresponds to this variable.
6.5 Semantics

To be able to compare DRT and TT, we now investigate the question as to how far the presented translation is able to preserve the semantics of DRT. Let $D$ be a DRS and $M$ a model, and assume that $D$ is verifiable on $M$. When we translate the DRS $D$ we get a type $T$ with $\mathbf{T}(D) = T$.

What can we say about the meaning of this type? The meaning of $T$ is, for a given agent, the ability to recognise an inhabitant of $T$. This ability in itself is independent of the question whether $T$ actually has an inhabitant, i.e. it is independent of the 'truth' of $T$. But in order to reproduce a classical semantics, we must be able to say something about $T$'s truth.

In principle every agent will judge the 'truth' of the given type for himself, using his own context. Contexts are not like objective world-models; they are knowledge states that are inherently partial and subjective, and represent some agent's personal convictions about the world. Consequently, the judgement of a given agent will not be universal; it will just reflect the judgement of that particular agent. Formally, the agent will judge a given type to be 'true' if he is indeed able to construct a realisation for the type, given the knowledge that he has. For the agent, the type $T$ 'is true' given his context $\Gamma$ if there exists an inhabitant $E$ such that $\Gamma \vdash E : T$.

6.5.1 Simulating 'classical' semantics

To reproduce a 'classical' semantics, like that of DRT, within the TT framework one has to assume the existence of an 'omniscient' agent $\Omega$, whose knowledge state $\Gamma_\Omega$ implicitly contains all the knowledge about the model $M$ in relation to which the truth of the DRSs is judged. This all-knowing agent $\Omega$ might be able to pronounce a judgement — an ultimate judgement, so to speak — that will carry enough weight to count as truth. If such an agent $\Omega$ is assumed, the judgement of this agent is equivalent to the semantics of DRT if:

$$\forall D : \text{DRS}.((\exists E.\Gamma_\Omega \vdash E : \mathbf{T}(D)) \Leftrightarrow (\exists g.(\Lambda, g) \in [D]))$$

The essential point that we need to consider, when investigating this equivalence, is the correspondence between the assignment $g$ and the inhabitant $E$. One should note that the inhabitant $E$ is in many ways more explicit than its counterpart $g$: $E$ does not only assign variables to entities, but also contains formal proofs that the substituted entities satisfy their requirements, i.e., that $g$ is indeed an assignment that verifies the DRS.

For this reason it is easy to construct $g$ if the realisation $E$ is given. As we remember, the inhabitant of a $\Sigma$-type is in essence a sequence. For instance,
an inhabitant $E$ of the type:

$$(\Sigma x : entity, (\Sigma q : (man \, x). (whistles \, x)))$$

such that

$$\Gamma \vdash E : (\Sigma x : entity, (\Sigma q : (man \, x). (whistles \, x)))$$

in some context $\Gamma$, will be the object: $\langle e_0, \langle e_1, e_2 \rangle \rangle$ where:

$$\Gamma \vdash e_0 : entity$$
$$\Gamma \vdash e_1 : (man \, e_0)$$
$$\Gamma \vdash e_2 : (whistles \, e_0)$$

Now each part of $E$ either corresponds to an inhabitant of type ‘entity’ (it corresponds directly to a discourse referent), or it corresponds to a condition. The parts that correspond to conditions are proof objects. The proof objects in question simply show that the given conditions do indeed hold, and why. They give a justification, so to speak, that shows that the given assignment to the discourse referents is correct. If all that we are interested in is the assignment of the top box of the DRS — and that is the case if we try to retrieve the assignment $g$ — we can simply leave out all these proof objects.

But to establish equivalence we must also show that it is possible to construct $E$ when $g$ is given. This is more difficult, as the assignment $g$ itself does not suffice to show that it really satisfies $D$ in $M$. All that the agent can find in $g$ is how the different discourse referents have been assigned, i.e. in TT terms, which are the entities that he has to use as inhabitants of the type ‘entity’. To derive the whole of $E$, the agent has to take into account not only $g$, but also the truth conditions on $g$ (cf. p. 162), i.e. the constraints on $g$ that are imposed by the fact that $\langle \Lambda, g \rangle \in [D]$. These constraints guarantee that the assignment $g$ is such that all conditions in $D$ are true in $M$. There are three kinds of conditions, of the following forms:

- $R(t_1, \ldots, t_n)$
- $t_1 = t_2$
- $D_a \Rightarrow D_b$

Given the conditions that hold for $g$, it is certain that the propositions of the form $R(t_1, \ldots, t_n)$ and $t_1 = t_2$ will hold in $M$. But in that case the agent $\Omega$ should, given his omniscience, also be able to prove that these propositions hold. If this is the case he can produce the needed proof objects.

The situation is more complicated in the case of complex conditions of the form $D_a \Rightarrow D_b$. Here any assignment of the discourse referents of $D_a$ must
be extendable to an assignment for the discourse referents of $D_b$. Now this boils down to the existence of functions in the model $M$ that yield entities that can be assigned to the discourse referents in $D_b$ given an assignment for the discourse referents of $D_a$.

Thus, if we want the semantics to be equivalent, the agent $\Omega$ should be able to construct these functions, and should also be able to construct the required proofs of their properties. As the reader can see, the requirements that we impose on $\Omega$ are rather idealistic. In particular, there will be problems if we allow the domain of discourse to be infinite.

**Problems in infinite models**

To get a feeling for what is required for the agent $\Omega$ to do his job when the domain of discourse is allowed to be infinite, consider what happens when we have a DRS $D_G$ for the following sentence, which expresses the Goldbach conjecture: “Every even number greater than 2 can be written as the sum of two primes.” There are now two different cases:

- If $G$ does not hold, the meaning of $D_G$ in DRT is $\{\}$. In that case, the agent $\Omega$ does not have to do anything. All that is required, is that $\Omega$ cannot prove $G$, which seems warranted.

- If $G$ holds, the meaning of the DRS $D_G$ is nothing but the set of pairs of identical assignments: $\{(g, g)\}$. For the ‘judgement’ semantics to be equivalent to DRT semantics, it is now required that $G$ is provable in the context $\Gamma_\Omega$.

This leads to problems. Though $G$ may well be true in standard arithmetic, the existence of a proof for $G$ cannot be guaranteed. On the contrary, we know that there are no complete formalisations of arithmetic. So for any ‘omniscient’ being $\Omega$ which is represented by some finite context $\Gamma_\Omega$ there exists some sentence $G$ for which the ‘judgement’ semantics is not equivalent to the ‘classical’ one.

We conclude that the two semantic notions are not equivalent if the domain of discourse can be infinite. However, it seems fair to say that the ‘classical’ semantics can be seen as a limiting case of a proof-oriented semantics.

Of course, in a practical setting, one does not have to worry about incompleteness. There is no reason to assume that the knowledge of an agent is complete. On the contrary, in real life all utterances are judged by agents that have only partial, and often even incorrect knowledge about a state of affairs. If our goal is to have a realistic and computational understanding of the way in which a normal agent handles a given discourse, the problems that we encounter when we try to emulate omniscient agents are of little or no importance.
6.5. SEMANTICS

6.5.2 The advantages of subjective judgement

In the realistic case, we are dealing with an agent $A$ that has a knowledge state given by $\Gamma_A$ and that is able to translate a given utterance to a type $T$.

This type $T$ must be well-formed on the knowledge state $\Gamma_A$, i.e. one must have:

$$\Gamma_A, x : T \vdash ok$$

Otherwise, the utterance simply does not make sense to the agent. If the utterance does make sense, it is immediately clear that, from the subjective viewpoint of the agent $A$, three situations may arise:

1. $$\exists E. \Gamma_A \vdash E : T$$

In this case, the agent $A$ was already informed about the content of the discourse, and this information is independently confirmed.

2. $$\exists E. \Gamma_A, x : T \vdash E : \bot$$

In this case, the agent is offered information that is inconsistent with his own knowledge.

3. Neither of the above. In this case the agent $A$ is offered information that is new to him.

One sees that the resulting classification is both more finegrained and more useful then a classification in terms of truth values. It is easy to imagine different ‘natural’ reactions that an agent might exhibit in the different situations:

- In the first case, the agent was already informed about the content of the discourse, and learns nothing new, so he might simply agree with the information offered.

- In the second case, he may well deny what has been said, and refuse to incorporate the new information, or he may try to revise his knowledge state in such a way that he can adopt this information while avoiding the inconsistency.

- In the third case, he may very well decide to adopt this new information, thus extending his context.
Note that in all these cases it is presupposed that the given utterance can be translated to a type that is well-formed on the context of A. This is not necessarily the case, since the translation of a given utterance could lead to a violation of type restrictions. This might happen when attempting to translate a sentence like: “Green ideas sleep furiously.” This sentence, which does not make sense to us, can also not be interpreted by the agent.

Of course, the present account is a rather crude simplification of a far more complicated process that involves two agents, the speaker and the hearer. So far, we have not indicated how the speaker may be able to produce utterances that can be comprehended by the listener, nor have we explained how the dialogue participants are able to use and extend a body of common knowledge. We will say more about some of these issues in the next chapter.

6.6 Conclusions and Discussion

In this chapter we have established a relation between the type-theoretical knowledge states that we propose in this thesis and Discourse Representation Theory, an existing theory of natural language interpretation. We have shown that it is possible to compute a mapping from Discourse Representation Structures to types. Under the mapping, the model-theoretic truth conditions of DRT translate in a natural way into satisfiability conditions for a type with respect to a given context. We found evidence supporting the view that DRSs and their (classical) semantics can be seen as a limit case of a more general theory. In this general theory, utterances relate to types that are well-formed on a given context. These types can be given a semantics based on satisfiability with respect to the knowledge of a given agent.

As DRT can be mapped into type theory, it stands to reason that natural language phenomena that can be modelled in terms of DRSs can also be modelled on type-theoretical contexts. Several authors have already shown how TT can be used in the treatment of presuppositions: [Krause, 1995] [Krahmer & Piwek, 1999] [Piwek & Krahmer, 2000]. In [Ahn & al., 1995] it is shown how TT can be used to handle scope ambiguities. More in general, in [Ranta, 1991], [Ranta, 1994] the author goes to great length arguing that Martin-Löf type theory is suitable as a semantic theory for natural language interpretation. Similar arguments for the use of type theory in natural language semantics are given in [Sundholm, 1986] and [Mönnich, 1985]. To us, this combined evidence suggests that theories of natural language interpretation may be combined with the sketched background of subjective type-theoretical knowledge states advocated here.
Chapter 7

Inter-agent communication

*The use of words is to be sensible marks of ideas, and the ideas they stand for are their proper and immediate signification.*

John Locke

In this chapter we investigate the transfer of information between two agents that both have a subjective knowledge state represented in type theory. We want to know whether and how information can be transferred between the agents, in spite of the subjective nature of their knowledge.

To this end, we first consider two agents that have knowledge states which are compatible, and that share a common vocabulary. As we will see, the common vocabulary enables them to communicate simple messages, but also constrains the meaningful messages that they can exchange.

Assuming the ability of agents to mark parts of their knowledge as private or shared, we identify basic operations that the agents may employ in the exchange of information. We then go on to show, through a number of examples, how these basic operations may indeed be used to achieve the transfer of information. The examples show how the agents are able to extend their common knowledge, how they can overcome some of the limitations imposed by their common vocabulary, and how the information that they can exchange depends on their common knowledge.

Finally, we briefly discuss the relevance of the above to the modelling of natural language communication, in particular in the context of man-machine dialogue.

7.1 Introduction

In the previous chapters the knowledge state of an agent has been modelled as a type-theoretical context. This knowledge is semantically grounded in the agent’s ability to recognise the inhabitants of certain types in the external world. Thus, the knowledge of the agent is both subjective and private.
Formally, the private nature of the knowledge of the agents is apparent in the circumstance that a type that is well-formed in the context of one agent is not well-formed in the context of an other agent.

If one accepts the private nature of knowledge, this raises the question how agents may ever be able to communicate. Clearly, if communication between two agents is to be successful, the knowledge states of the agents must be compatible. The agents have to share (part of) their view on the world, understanding it through similar concepts. They have to distinguish similar objects, properties and events. If they do so, there can be a common language having words that denote these common ideas. This language may provide the connection between the private concepts of the agents, and goes some way towards explaining how the agents may be able to get their ideas across.

When we consider ‘public’ languages that are used by sizeable groups of agents, and in a wide range of situations, then it may be noted that the vocabulary of these languages only supports concepts that may be encountered in a variety of situations, and which are common to most speakers of the language. Accordingly, public languages tend not to have words for most of the individual objects\(^1\) that we encounter in daily life.

Nevertheless, speakers that interact in a common language succeed quite well in referring to the diverse objects and events that are relevant to them, without being forced to extend their language with situation-specific references. Also, the ability to exchange ideas typically increases as the dialogue goes on, which suggests that they develop some common knowledge during the communication process that functions as a resource that supports the successful exchange of information.

In this chapter we model these processes in a very simple way. We add some extra structure to the knowledge states of agents, to take their common knowledge into account, and consider computational interpretation mechanisms that allow information exchange between agents with private knowledge states. These mechanisms are of interest, as they:

- may form part of implementations that support the exchange of ideas between such agents, or between such an agent and a human;
- may help us to understand how the lack of shared name spaces affects the way in which agents refer to objects.

\(^1\)The reader should note that many artificial languages, like progranmming languages, offer mechanisms for the declaration of variables through which the language is able to cope with the above problem. The language user can effectively extend the language with ‘private’ expressions that enable him to refer to specific objects in the given situation. This may be one of the reasons why it is so difficult to reuse software, in the absence of mechanisms for data-hiding.
7.2. SETTING THE SCENE

We focus on the simple case where two agents communicate about their common environment, using examples where these agents discuss an electron microscope, the situation of the ‘DenK’ project [Bunt, 1998].

7.2 Setting the scene

We consider a situation with two agents, $A$ and $B$, that want to exchange information. The knowledge states of the agents are modelled by means of type-theoretical contexts. Each agent $\alpha$ has a knowledge state $\Gamma_\alpha$. Within $\Gamma_\alpha$ the various concepts that $\alpha$ uses to understand the surrounding world can be found.

The knowledge states of the agents are compatible in the sense that they recognise similar concepts. We assume that the agents speak a common language that functions as a vehicle of communication.

The precise nature of this language does not concern us here, we only assume that the language offers a certain vocabulary which allows the agents to verbalise their private concepts. From a formal point of view, we assume that for each agent ($\alpha$) there exists a partial mapping $\mathcal{V}_\alpha = W$ between variables $\mathcal{V}_\alpha$ that occur in his knowledge state and words in the vocabulary $W$ of the shared language.

7.2.1 The common vocabulary

Take a situation where $A$ and $B$ have a common vocabulary of English words. The words in the vocabulary correspond for a given agent $\alpha$ with variables that are declared in his knowledge state $\Gamma_\alpha$, in accord with the mapping $(\mathcal{V}_\alpha \equiv W)$. The situation is represented in the following table:

<table>
<thead>
<tr>
<th>$\Gamma_A$</th>
<th>$\mathcal{V}_A \equiv W \equiv \mathcal{V}_B$</th>
<th>$\Gamma_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 : \ast_t,$</td>
<td>$x_1 \equiv \text{bundle} \equiv y_{17}$</td>
<td>\ldots</td>
</tr>
<tr>
<td>$x_2 : x_1 \rightarrow \ast,$</td>
<td>$x_2 \equiv \text{primary} \equiv y_{22}$</td>
<td>$y_5 : \ast_t,$</td>
</tr>
<tr>
<td>$x_3 : \ast_t,$</td>
<td>$x_3 \equiv \text{lens} \equiv y_5$</td>
<td>$y_6 : y_5,$</td>
</tr>
<tr>
<td>$x_4 : x_1 \rightarrow x_3 \rightarrow \ast,$</td>
<td>$x_4 \equiv \text{enter} \equiv y_9$</td>
<td>\ldots</td>
</tr>
<tr>
<td>$x_5 : x_1,$</td>
<td>$\ldots,$</td>
<td>$y_{17} : \ast_t,$</td>
</tr>
<tr>
<td>$x_6 : x_3,$</td>
<td>$y_{18} : y_{17}$</td>
<td>$y_{19} : y_{17} \rightarrow y_5 \rightarrow \ast,$</td>
</tr>
<tr>
<td>\ldots</td>
<td>$\ldots,$</td>
<td>\ldots</td>
</tr>
<tr>
<td></td>
<td>$y_{52} : y_{17} \rightarrow \ast,$</td>
<td>\ldots</td>
</tr>
</tbody>
</table>

In this table, which lists a part of the knowledge of $A$ and $B$, we see that agents $A$ and $B$ have a shared vocabulary that contains at least the words ‘bundle’, ‘lens’, ‘primary’ and ‘enter’. The type variables corresponding to
these words are declared in both contexts $\Gamma_A$ and $\Gamma_B$. Note that $A$ and $B$ have different type variables that correspond to the same word: in $A$’s knowledge state the concept ‘lens’ is mapped to the type $x_3$, in $B$’s knowledge state it is mapped to the type $y_5$.

What is interesting in the light of our concerns is that the mapping between variables and words is not a complete mapping; it is subject to an important constraint:

- words must refer to concepts and ideas which are useful to the language community as a whole: there can be no words that relate to concepts that are only relevant to a single agent, or within a single situation that is being discussed.

A consequence of this constraint is that there are hardly any words that may be used to refer to individual objects in the world: words can only refer to individuals if these are relevant to a large part of the language community. This is important, as the limitations of the common vocabulary constrain the ideas that the agents can verbalise.

### 7.2.2 Coding and decoding

Consider an idea that is meaningful to $A$. This corresponds to an extending segment $\Delta_A$ of his knowledge state $\Gamma_A$. To send this idea to $B$, $A$ must encode the segment $\Delta_A$ into a public message, using the common vocabulary. This message is sent to $B$ who subsequently decodes it. If the communication is to be successful, the result of decoding must be meaningful to $B$, i.e. it must be an extending segment $\Delta_B$ of $B$’s context $\Gamma_B$.

To model this, we will assume that both encoding and decoding will be based on the common vocabulary. The agent $A$ employs the mapping $\mathcal{V}_A = W$ to encode a segment $\Delta_A$, using the following trivial encoding procedure: To encode the segment, $A$ simply substitutes in the segment $\Delta_A$ the words of the vocabulary for the variables that they are related to.

The result of encoding is sent to the recipient ($B$), that can try to decode it. We assume that decoding is the inverse of encoding, except that one now has to use the recipient’s mapping, $\mathcal{V}_B = W$, in the direction from words to types. Under the assumption that the mapping between words and types is one to one, encoding and decoding are rather trivial procedures, that do not, by themselves, add anything of interest.

The main reason why coding and decoding are nonetheless considered here, is that the necessity to encode and decode extending segments of contexts clarifies how the common vocabulary constrains the segments that can be communicated. As there are variables that do not correspond to words, there are many segments $\Delta_A$ which are meaningful to $A$ but which cannot be encoded in such a way that they may be successfully decoded by $B$. 


7.2. SETTING THE SCENE

The formal reason is that it must be guaranteed that any message decoded by B no longer contains any variables that are declared (i.e. bound) in the context $\Gamma_A$. To ensure this, all these variables must disappear in the encoding process. This can only be the case if A has words for all variables that occur free in the segment $\Delta_A$ it wants to communicate. This simple requirement therefore puts strong limitations on the segments that A can use to communicate with B. Segments that meet this requirement we call codeable. Formally, codeable segments can be characterised by the following definitions:

**Definition 1** A variable $z$ occurs free in a segment $\Delta$, $\Delta \equiv x_1 : T_1, \ldots, x_n : T_n$ iff $z$ occurs free in $T_i$ $(1 \leq i \leq n)$ and there is no statement $x_j : T_j$ with $1 \leq j < i$ such that $z \equiv x_j$.

**Definition 2** A segment $\Delta_A$ is codeable iff for all variables $z$ occurring free in $\Delta_A$ there is a word $w$ in $W$ such that $z \models_A w$.

Coding a codeable segment yields a public message. This message can subsequently be decoded, and yields a extending segment of the context of the hearer.

In the sequel we will investigate how agents can exchange information through these codeable segments. To enhance clarity of exposition, we will from now on rewrite all contexts and extending segments for a given agent modulo the coding of this agent itself, assuming a vocabulary of English words.

### 7.2.3 Common knowledge

When A and B are exchanging information the agents are not only extending their individual knowledge. They also extend their common knowledge. As is apparent from the literature, the common knowledge of the agents plays a crucial role in the communication process. See for instance [Stalnaker, 1974, Stalnaker, 1978], [Clark & Marshall, 1981].

To model common knowledge and its dynamic development, we will elaborate on the simple model of type-theoretical knowledge states related to a common vocabulary that we have used so far. We add some extra structure to the knowledge states, through which two kinds of knowledge are distinguished:

1. private knowledge: knowledge that is only known to the agent himself.

2. common knowledge: that part of the agent’s knowledge that he assumes to be shared with his partner.
Formally, the private knowledge of an agent $\alpha$ is modelled by a context $\Gamma_\alpha$ which contains his entire current knowledge. In order to model the common knowledge, certain declarations in $A$'s knowledge are marked in some way, and it is assumed that within $\Gamma_\alpha$ the agent $\alpha$ can distinguish a (sub)context $\Psi_\alpha$, with $\Psi_\alpha \subseteq \Gamma_\alpha$, which contains all knowledge that, according to $\alpha$, is shared. Of course, it can not be guaranteed that the knowledge in $\Psi$ is shared, but $\Psi$ represents an agent's approximation of the common knowledge.

Both $\Gamma_\alpha$ and $\Psi_\alpha$ are well-formed type theoretical contexts in their own right. The common context $\Psi_\alpha$ is 'a part of' the private context $\Gamma_\alpha$. Formally, this relation can be defined in a straightforward way:

**Definition 3** Given two legal contexts $\Psi$ and $\Gamma$, $\Psi$ is a part of $\Gamma$, notation $\Psi \subseteq \Gamma$, iff

1 for all statements of the form $x : T$ occurring in $\Psi$ either $x : T$ occurs in $\Gamma$ or a definition $x = E : T$ occurs in $\Gamma$.

2 all statements of the form $x = E : T$ occurring in $\Psi$ occur also in $\Gamma$.

Under this 'part of'-relation, every definition in $\Psi$ must occur in $\Gamma$ (2), but declarations in $\Psi$ may be replaced by definitions in $\Gamma$ (1). This will be of use in Sect. 7.3, where it allows us to 'anchor' shared information in $\Psi_\alpha$ to private information in $\Gamma_\alpha$.

This distinction between common and private knowledge also gives rise to a somewhat finer-grained account of reasoning. Since both $\Gamma_\alpha$ and $\Psi_\alpha$ are legal contexts, new statements can be constructed on either context using the derivation rules. Hence we can model the agent reaching 'private conclusions' in reasoning with private information ($\Gamma_\alpha$) and 'common conclusions' in reasoning with common information ($\Psi_\alpha$). Information that is shared with another agent is also privately available, as reflected in the inclusion $\Psi_\alpha \subseteq \Gamma_\alpha$. This inclusion guarantees that any statement derivable on an agent's common context ($\Psi_\alpha$) is also derivable on his private context ($\Gamma_\alpha$), but not the other way round.

**Example**

Consider an agent $A$, that has a knowledge state $\Gamma_A$ concerning the workings of an electron microscope, as in the case in the DenK project. In order to illustrate matters, it suffices to take a small sample from this context. (The original context of the DenK system contains hundreds of clauses.)

In this fragment, types are declared for 'lenses' an (electron) 'bundle', and a 'gun'. Bundles can 'enter' certain parts of the microscope 'bundle', and there are some facts and rules that enable the agent to
construct justifications of propositions. For example, in the context below, the proposition \( x_5 \) enters \( x_6 \) has the justification abbreviated as \( 'M' \).

\[
\begin{array}{|l|}
\hline
\Gamma_A \\
\hline
\ldots \\
\text{bundle} : *_t, \\
\text{primary} : \text{bundle} \rightarrow *, \\
\text{lens} < \text{thing} : *_t, \\
\ldots \\
\text{enter} : \text{bundle} \rightarrow \text{thing} \rightarrow *, \\
\text{emit} : \text{thing} \rightarrow \text{bundle} \rightarrow *, \\
\ldots \\
x_5 : \text{bundle}, \\
\ldots \\
x_6 : \text{lens}, \\
x_7 : (\text{primary } x_5) \\
\text{condensor} : \text{lens} \rightarrow *, \\
x_8 := M : (\text{enter } x_5 \ x_6) \\
x_9 : \text{lens} \\
\text{gun} < \text{thing} : *_t, \\
x_{48} := N : (\text{condensor } x_6), \\
x_{50} : \text{gun}, \\
\text{below} : \text{thing} \rightarrow \text{thing} \rightarrow *, \\
x_{51} : (\text{below } x_{50} \ x_6), \\
\hline
\end{array}
\]

Given this knowledge state \( \Gamma_A \), the precise shared context \( \Psi_A \) depends of course on the communication partner \( B \) that \( A \) wants to address. In general, the relationship between the private and the common context can be characterised as follows:

- Declarations of variables that correspond to words in the shared vocabulary can be assumed to be common, as it is reasonable to assume that both agents share a concept for which they both have a word.

- Depending on the situation, \( A \) and \( B \) may share more than just their vocabulary, even at the beginning of a dialogue. There may be certain general knowledge which they can correctly assume to share with their partner, or they may share certain information as a result of a previous conversation. This information will then also be represented in their common contexts.

- Selecting all the common elements in \( A \)'s private context, a well-formed context must result. This expresses the fact that certain knowledge can only be shared if presuppositions of this knowledge are also shared. For instance, given the above context \( \Gamma_A \), one can share the declaration: \( x_{48} : (\text{condensor } x_6) \) only if one also shares the declaration: \( x_6 : \text{lens}. \)
As there are two agents, there are also two versions of the common context. The context $\Psi_A$ formalises the common knowledge from the viewpoint of the agent $A$. A similar context $\Psi_B$ does so for the agent $B$. In communication both agents can only refer to entities that occur in this common context. However, this context can be extended during the communication process.

In the example at hand, the common knowledge as seen from the viewpoint of the agent $A$ may at a certain point in time be represented by the following context:

\begin{verbatim}
<table>
<thead>
<tr>
<th>\Psi_A</th>
</tr>
</thead>
<tbody>
<tr>
<td>bundle : (*_{t,}</td>
</tr>
<tr>
<td>primary : bundle \rightarrow *,</td>
</tr>
<tr>
<td>lens : (*_{t,}</td>
</tr>
<tr>
<td>\ldots,</td>
</tr>
<tr>
<td>enter : bundle \rightarrow thing \rightarrow *,</td>
</tr>
<tr>
<td>emit : thing \rightarrow bundle \rightarrow *,</td>
</tr>
<tr>
<td>x_5 : bundle,</td>
</tr>
<tr>
<td>x_6 : lens ,</td>
</tr>
<tr>
<td>x_8 : lens</td>
</tr>
<tr>
<td>condensor : lens \rightarrow *,</td>
</tr>
<tr>
<td>x_{48} : (condensor x_{6}),</td>
</tr>
<tr>
<td>gun : (*_{t,}</td>
</tr>
<tr>
<td>x_{50} : gun,</td>
</tr>
</tbody>
</table>
\end{verbatim}

The common knowledge is the backbone of the communication between $A$ and $B$. For agents who share all the concepts in their vocabulary, every codeable extending segment of their own context is also an extension of the common knowledge of the agents, and, a fortiori, of the knowledge state of each agent itself. The messages that the agents will exchange will be extending segments of their common knowledge.

### 7.2.4 Messages

The previous sections have shown how the common vocabulary allows content that is privately meaningful to agent $A$ can be coded in a public message that can be decoded by the agent $B$. Messages, i.e. the objects that communicating agents exchange, are not just expressions of content, however. A message is always the expressions of a certain communicative intention: the sender wants the addressee not just to understand the content, but also to do something with it. If $A$ wants to provide $B$ with some information $p$, for instance, then $A$’s message will indicate that $B$ is expected to add the content $p$ to his knowledge state (more specifically, assuming that $A$ is sincere, i.e. that $A$ himself believes that $p$, agent $B$ should add this to the knowledge that
he believes to be shared with $A$). But if instead $A$ wants to know whether $p$ is the case, then his message to $B$ will indicate that $B$ is expected to try to construct a proof of $p$ (and tell $A$ whether he succeeds). In other words, $A$'s messages will encode both a certain content and an intention of how $B$ should operate with that content.

These two aspects of a message are familiar in modern action-based approaches to natural language communication, like Communicative Activity Analysis (CAA) [Allwood, 1976], [Allwood & al., 1992], [Allwood, 1994] as well as Dynamic Interpretation Theory (DIT) [Bunt, 1989], [Bunt, 1990], [Bunt, 1994], [Bunt, 2000]. In DIT, for instance, the communication process between two agents is modelled as consisting of communicative actions, called ‘dialogue acts’, which are characterised semantically by their intended effect on the context, in particular on the addressee’s knowledge state. A dialogue act has two semantic components, a semantic content and a communicative function, defined as the information that the sender is introducing into the context and the way in which the addressee is intended to handle that information. When natural language is used for communication, we see that a speaker uses for instance word order and intonation for encoding communicative functions, see e.g. [Beun, 1990].

In this thesis, we do not attempt to model natural language communication, but we do have to take into account that messages contain both content and function. To this end we combine segments with simple annotations (‘tags’) that instruct a receiving agent what to do with the type-theoretical content of the message. The tags correspond to the various options that the agents have in processing a given segment.

### 7.2.5 Processing of segments

An agent that has to process a given segment faces several options in doing so. As we have seen in chapter 2, any extending segment $\Delta$ of a context $\Gamma$ can be related to this context in two ways:

- The segment $\Delta$ can be regarded as a hypothesis extending $\Gamma$.
- The segment $\Delta$ can be regarded as a specification for which a realisation is to be constructed in $\Gamma$.

We call the former interpretation segments ‘positive’, the latter ‘negative’.

Under the positive interpretation, a segment will extend a knowledge state with new information. The new information is simply stored as a set of additional ‘hypotheses’ or ‘assumptions’ (all justifications in the segment are atomic).

The processing of segments under the negative interpretation is somewhat more complicated. In this case, the agent has to find justifications (objects
and proofs) in his own current context to replace the dummy inhabitants of the statements in the extending segment. In other words, the agent should construct a ‘realisation’ for the extending segment in his original context. We recall the definition of a realisation given in chapter 2, definition 9.

Definition 4 let $\Delta \equiv x_1 : T_1, \ldots, x_n : T_n$ be an extending segment of $\Gamma$, and let
\begin{align*}
\Gamma &\vdash D_1 : T_1 \text{ and } \\
\Gamma &\vdash D_2 : T_2[x_1 := D_1], \text{ and } \\
\Gamma &\vdash D_3 : T_3[x_1 := D_1, x_2 := D_2], \text{ and } \\
&\ldots \text{ and } \\
\Gamma &\vdash D_n : T_n[x_1 := D_1, \ldots, x_{n-1} := D_{n-1}] \text{ then we call } \\
\Delta^* &\equiv x_1 = D_1 : T_1, \ldots, x_n = D_n : T_n \text{ a realisation of } \Delta \text{ in } \Gamma \text{ under the substitution } [x_1 := D_1, \ldots, x_n := D_n].
\end{align*}

Once a realisation is found, the agent appends the realisation $\Delta^*$ to the context. In this way he has used the variables in the segment as selective ‘hooks’ that connect to particular inhabitants of this context. As we will see, the ability to do so is crucial to successful information exchange.

Another choice that plays a role in the interpretation of segments is a consequence of the fact that we choose to model the knowledge state of an agent by a context that has two parts: a private context, and a common context. As the common context is a part of the private context, any extending segment of the common context is also an extending segment of the private context. Therefore the agent can, in principle, choose to process the segment on either context.

## 7.3 Communication

As we have seen in the previous section, the agent has the ability, for a given segment $\Delta$:

- to extend his private context $\Gamma_\alpha$ or his shared context $\Psi_\alpha$ with the segment $\Delta$.

- to extend his private context $\Gamma_\alpha$ or his shared context $\Psi_\alpha$ with a realisation $\Delta^*$ that he constructed on this context.

In this section we investigate how the agents can use these abilities to exchange information, using messages that consist of ‘tagged’ encoded segments.
7.3. COMMUNICATION

7.3.1 The situation

Consider the agent $A$, that has a knowledge state $\Gamma_A$ concerning the workings of an electron microscope, as given in section 7.2.3.

<table>
<thead>
<tr>
<th>$\Gamma_A$</th>
</tr>
</thead>
<tbody>
<tr>
<td>\ldots, \quad bundle : \ast_t, \quad primary : bundle \rightarrow \ast, \quad lens &lt; thing : \ast_t, \quad \ldots, \quad enter : bundle \rightarrow thing \rightarrow \ast, \quad emit : thing \rightarrow bundle \rightarrow \ast, \quad \ldots, \quad x_5 : bundle, \quad \ldots \quad x_6 : lens, \quad x_7 : (primary x_5) \quad condensor : lens \rightarrow \ast, \quad x_8 := M : (enter x_5 x_6) \quad x_9 : lens \quad gun &lt; thing : \ast_t, \quad x_{48} := N : (condensor x_6), \quad x_{50} : gun, \quad below : thing \rightarrow thing \rightarrow \ast, \quad x_{51} : (below x_{50} x_6),</td>
</tr>
</tbody>
</table>

We assume that agent $A$ communicates with an agent $B$, and that the agents have the following shared knowledge:

<table>
<thead>
<tr>
<th>$\Psi_A$</th>
<th>$\Psi_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>bundle : \ast_t, \quad primary : bundle \rightarrow \ast, \quad lens : \ast_t, \quad enter : bundle \rightarrow lens \rightarrow \ast, \quad x_5 : bundle, \quad x_6 : lens, \quad condensor : lens \rightarrow \ast \quad x_7 : (primary x_5) \quad x_{48} : (condensor x_6) \quad gun : \ast_t</td>
<td>lens : \ast_t, \quad y_6 : lens, \quad condensor : lens \rightarrow \ast \quad bundle : \ast_t, \quad y_{18} : bundle \quad enter : bundle \rightarrow lens \rightarrow \ast, \quad y_{52} : (condensor y_6) \quad gun : \ast_t \quad primary : bundle \rightarrow \ast, \quad y_{30} : (primary y_{18})</td>
</tr>
</tbody>
</table>

The agents $A$ and $B$ can use tagged messages to exchange information. We consider two basic acts of communication: providing information, and seeking information.
7.3.2 Providing information

Suppose that agent $A$ wants to provide information to agent $B$. In fact, $A$ wants to tell $B$ that the commonly known bundle, to which $A$ himself refers as $x_5$, enters some lens, with whose identity $A$ is presently not concerned. Formally, $A$ wants to extend $B$’s knowledge through an extension of the knowledge that he shares with $B$. The segment that $A$ wants to add to the common context, is:

$$ l : lens, q : (enter x_5 l) $$

Agent $A$ takes this to be fact, he has a realisation for it on his private context:

$$ l := x_5 : lens, q := M : (enter x_5 l) $$

However, the segment $l : lens, q : (enter x_5 l)$ is not codeable, as it contains the free variable $x_5$. In an attempt to capture the free variable, $A$ could choose to communicate the slightly longer segment:

$$ b : bundle, l : lens, z : (enter bl) $$

This segment is codeable, and indeed meaningful to $B$, as it is an extending segment of $B$'s common context $\Psi_B$, and, a fortiori, of $B$'s own context $\Gamma_B$. Unfortunately, however, the segment does not convey the idea that $A$ wanted to communicate: the identity of the bundle is now lost.

But as the bundle $x_5$ is shared, $A$ is able to communicate its identity. To do so, $A$ must present $B$ with a description of the bundle that he wants to refer to, a description that $B$ can decode.

To convey his thought, $A$ must use a more complex message, consisting of two parts: the first part of this message contains the description of the bundle and the second part contains the information about this bundle that $A$ wants to convey. These parts have to be processed by $B$ in different ways. The first part of the message is a description, for which $B$ has to provide a realisation, the second part consists of new information that the agent $B$ should add to his knowledge state. The description should be labelled in such a way that $B$ interprets it negatively, and indeed treats it as a description for which he has to find a realisation on the common knowledge. For the sake of definiteness, we assume that the agents use the following tags:

- $+/-$ to denote the positive or negative polarity of the segment.
- $\Psi/\Gamma$ to denote whether the segment has to be processed on the common or the private context.
7.3. COMMUNICATION

Using these tags, $A$ can send the following message:

$$[b : \text{bundle}, p : (\text{primary } b)]^- [l : \text{lens}, q : (\text{enter } b l)]^+ \quad (7.1)$$

Note, that both parts of the message correspond to (de)codeable segments. The tags show that the message has a positive and a negative part. The first, negative, part ($\mu_1$) corresponds to a description of the primary bundle:

$$[b : \text{bundle}, p : (\text{primary } b)]^- \quad (7.2)$$

As the tags indicate, $B$ has to find a realisation for $b$ and $p$ on his common context $\Psi_B$ for the segment $\mu_1$. Assuming that $A$’s use of the description was appropriate, $B$ will find in his common context:

- an object, say $y_{18}$, representing the bundle.
- a proof ($y_{30}$) that $y_{18}$ is the primary bundle.

After extending his common context with the realisation:

$$b := y_{18} : \text{bundle}, p := y_{30} : (\text{primary } b)$$

$B$ processes the second part of the message. This part ($\mu_2$) contains the proposition asserted by $A$ that for some lens $l$ the primary bundle enters $l$:

$$[l : \text{lens}, q : (\text{enter } b l)]^+ \quad (7.3)$$

The second part of the message carries tags that instruct $B$ to interpret it positively; $B$ is supposed to ‘accept’ the information in the segment $\mu_2$, thereby extending his context. The first declaration in $\mu_2$ introduces a new lens ($l$) into $B$’s common context. The second declaration introduces a new ‘piece of evidence’ into the common context, a proof object ($q$) for the proposition that $b$ enters $l$. Note that, after processing $\mu_1$, the variable $b$ that occurs free in $\mu_2$ is bound in $B$’s extended common context by the definition $b := y_{18} : \text{bundle}$. The processing of the message as a whole therefore updates the common context of $B$ in two steps:

1. $\Psi_B \Rightarrow \Psi_B, \mu_1^*$ where $\mu_1^*$ is a realisation of $\mu_1$ in $\Psi_B$.

2. $\Psi_B, \mu_1^* \Rightarrow \Psi_B, \mu_1^*, \mu_2$.

It should be noted that the reaction of agent $B$ in this example is the simplest or ‘most cooperative’ one possible; he adds the information provided by $A$ without questioning it in any way. Depending on factors in the dialogue situation not considered here, a more ‘cautious’ reaction could be appropriate.

\[ \text{For instance, the DenK system will not accept assertions about an electron microscope made by the user, because it is considered to be an expert in this area.} \]
The sending agent has certain expectations about the alternative ways in which the receiver can deal with a given message. For instance, he may expect the receiver to either accept a given assertion, or to contest it, or to signal presupposition failure. He has to distinguish between these cases, in order to update his own knowledge accordingly.

For instance, when A learns that B has understood the message and accepted it, he has to ‘mimic’ B’s actions in his own version of the common context, in order to assure that both versions of the common context remain compatible. As we described above for B, this results in an extension of the common context: \( \Psi_A \Rightarrow \Psi_A, \mu_1, \mu_2 \). After these actions, the common context(s) of A and B look as follows:

<table>
<thead>
<tr>
<th>( \Psi_A )</th>
<th>( \Psi_B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>bundle : (<em>_t, ) primary : bundle ( \rightarrow ) ,</em></td>
<td>lens : (*_t, ) y_6 : lens,</td>
</tr>
<tr>
<td>lens : (<em>_t, ) enter : bundle ( \rightarrow ) lens ( \rightarrow ) ,</em></td>
<td>condensor : lens ( \rightarrow ) ,*</td>
</tr>
<tr>
<td>x_5 : bundle, x_6 : lens, condensor : lens ( \rightarrow ) ,*</td>
<td>bundle : (*_t, ) y_8 : bundle</td>
</tr>
<tr>
<td>x_7 : (primary x_5) x_8 : (condensor x_6)</td>
<td>enter : bundle ( \rightarrow ) lens ( \rightarrow ) ,*</td>
</tr>
<tr>
<td>gun : (*_t</td>
<td>y_30 : (primary y_18)</td>
</tr>
<tr>
<td>b_A := x_5 : bundle</td>
<td>b_B := y_18 : bundle</td>
</tr>
<tr>
<td>p_A := x_7 : (primary x_5)</td>
<td>p_B := y_30 : (primary y_18)</td>
</tr>
<tr>
<td>l_A : lens</td>
<td>l_B : lens</td>
</tr>
<tr>
<td>q_A : (enter b_A l_A)</td>
<td>q_B : (enter b_B l_B)</td>
</tr>
</tbody>
</table>

The agent A will know more about the bundle \( b_A \) and the lens \( l_A \) then is apparent from the common context. He has certain private information that is connected to and goes beyond the common information. In particular, as we have seen, his original assertion rests on the fact that A has a realisation of \( \mu_2 \) on his private context \( \Gamma_A : \mu_2^* \equiv l := x_6 : lens, q := M : (enter b l) \). This realisation expresses in which respects A knows more in his private context than in his common context: in \( \Gamma_A \) he has additional information about the identity of the lens \( x_6 \) which the bundle enters, information that is not shared, and which cannot be derived about the newly introduced lens \( l_A \) in \( \Psi_A \). Moreover, in \( \Gamma_A \) he has a structured proof \( M \) rather than an unstructured ‘dummy’ \( q_A \) in \( \Psi_A \). Agent A can connect his ‘private’ justifications to his ‘common’ justifications by updating his private context with the realisation:

\(^3\)To judge whether this is the case, A is dependent on the feedback given by B.
(\(\Gamma_A \Rightarrow \Gamma_A, \mu_1^A\)). In this way, \(x_6\) is linked to \(l_A\) and \(M\) to \(q_A\). Through these links, \(A\) is able to relate further information about (his own version) of the common lens \(l_A\) to the particular lens \(x_6\) in his private context. Note that \(b_A\) was already linked to \(A\)'s representation of the primary bundle by the update of \(\Psi_A\) with \(\mu_1^A\).

### 7.3.3 Seeking information

Suppose that \(B\) wants to obtain further information from \(A\). \(B\) does not know which lens the primary bundle enters, but he may suspect this lens to be a condensor lens. To check this, \(B\) can use the following message:

\[
[l : \text{lens}, b : \text{bundle}, q : (\text{enter b l})]_{\Psi} \rightarrow [c : (\text{condensor l})]_{\Gamma} \tag{7.4}
\]

The first part of the message (\(\mu_5\)) is tagged in a familiar way, and again functions as a description:

\[
[l : \text{lens}, b : \text{bundle}, q : (\text{enter b l})]_{\Psi} \tag{7.5}
\]

It will be processed by \(A\) yielding an update of \(\Psi_A\), a realisation for the lens that the primary bundle enters, binding 'T' in the right way. The tags for the second part of the message (\(\mu_4\)) are different:

\[
[c : (\text{condensor l})]_{\Gamma} \tag{7.6}
\]

This segment has a negative polarity, instructing agent \(A\) to view it as a specification. Also, \(\mu_4\) has to be processed on the private context. In other words, \(A\) is supposed to construct a realisation on \(\Gamma_A\) for the segment \(\mu_4\); he has to find a proof that the lens is a condensor lens. Inspecting the knowledge state \(\Gamma_A\), we see that \(A\) is able to construct such a realisation: \(c := x_{48} : (\text{condensor l})\). The update of \(A\)'s private context with this realisation, \(\Gamma_A \Rightarrow \Gamma_A, \mu_4^A\), brings \(A\) in the position where he (privately) possesses all the information needed to answer \(B\)'s question.

The proof that \(A\) finds must be 'strictly private' in the sense that it is available on \(\Gamma_A\) but not on the subcontext \(\Psi_A\): if the realisation could also be constructed on \(\Psi_A\), a realisation could also be constructed on \(\Psi_B\) and then \(B\)'s request for information would be superfluous.

If \(A\) wishes to answer \(B\), he has to give some indication whether he has been able to construct such a realisation. In the case at hand he should be giving some positive feedback\(^4\). One way of doing this is that \(A\) turns his private extension into a common one, by sending the following confirmative message:

\[
[l : \text{lens}, b : \text{bundle}, q : (\text{enter b l})]_{\Psi} \rightarrow [c : (\text{condensor l})]_{\Psi}^+ \tag{7.7}
\]

\(^4\)That such feedback is expected should be indicated in the original message.
This message, when processed by $B$, will achieve the required state of common knowledge.

### 7.4 Applying the mechanisms

As these examples show, two agents with knowledge states represented in type theory can exchange information through messages that consist of 'extending segments' complemented with pragmatic annotations. In this way, they are able to gradually extend their common knowledge. The agents interpret the diverse extending segments on their knowledge states (type theoretical contexts) in accord with the given annotations. As these contexts are subject to change, the agents can interpret identical messages in different ways, dependent on their current knowledge state. So the resulting communication mechanism is inherently context-dependent. As it is also a computational mechanism, it may be applied, for instance in man-machine interfaces in which the user communicates with an artificial agent.

The selling point of such a type of interface is that it may bridge the gap between the knowledge of the user and the knowledge of the agent in an incremental way, as the common knowledge of user and agent is extended. If the agent really adheres to the constraint that all messages to the user correspond to codeable segments, the agent will be forced to adapt its responses to the actual knowledge of the user, as, for instance, it can only refer to objects that are mutually known.

Unfortunately, however, there is a practical problem that complicates the construction of such an interface: one needs a public language that can be used to carry the annotated segments across. This must either be a natural language, such as English, or some artificial language, which may be supported by such such devices as menu’s and dialogue boxes.

In the DenK project [Bunt, 1998] we have gained some experience with the first approach. In this project a man-machine interface has been constructed, in which a user and a software agent both have access to a given domain. (The domain is a simulated electron microscope.) The artificial agent has background knowledge about the domain that is represented in type theory. The user and the agent can communicate using a small fragment of English. Utterances within the fragment are interpreted in terms of annotated extending segments.

To interpret user utterances, the system uses a HPSG parser for a fragment of English, see: [Verlinden, 1999]. The parsed utterances are subsequently interpreted in terms of tagged segments, using a variety of pragmatic tags, see: [Kievit, 1998, pp. 63-67]). These tags code information about the use of determiners and demonstratives, modal auxiliaries, wh-elements, inversion, etc. in the original utterance.
7.4. APPLYING THE MECHANISMS

Guided by these tags, the system interprets the various utterances against the background of the knowledge state and given the (observable) state of the external reality. [Beun & Kievit, 1996], [Kievit, 1998].

Finally, on the basis of the information about modal auxiliaries, inversion, etc. the system determines how to respond to the various utterances, providing information or executing simple tasks. See [Beun & Piwek, 1997], [Piwek, 1995], [Piwek, 1998].

Thus, the interpretation mechanisms within the DenK system that underlie its natural language understanding capabilities are similar to the mechanisms that have been presented in section 7.3, though they are more complex for a number of practical reasons. In particular:

- The representation of the knowledge state of the agents is more intricate, and incorporates extra information about the dialogue history, reflecting the fact that the order in which entities are introduced in a dialogue is relevant, for instance to the resolution of anaphora.

- The interpretation mechanisms distinguishes three contexts (i.e. private, common and observable, [Kievit, 1998, pp. 72-76]) instead of two, reflecting the ability of both user and agent to observe the shared external reality.

- The system does not only respond to requests for information, but also to requests for action.

Experience with the DenK system shows that the approach is feasible in principle. It allows user and agent to communicate about the shared domain, using a small fragment of natural language. However, at present, the system suffers from important limitations that make its use in real applications impractical. One problem is that the system can only interpret a small fragment of English. Users have no way of knowing this fragment, and tend to produce utterances that the system cannot handle, which leads to disappointments. A second, more serious limitation is that the system is purely reactive: it only responds directly to requests made by the user, and cannot take any initiative. The reason for this is that the system does not have any explicit internal representation of intentions, neither for its own intentions, nor for those of the user. Accordingly, the system cannot reason about intentions or actions, and cannot take any initiative.

This last limitation is so serious, as successful dialogue, which is normally a cooperative activity, is shaped by the ability of the participants to infer the goals of their dialogue partners, both in the domain and in the dialogue itself. In fact, in approaches to communication that are based on speech act theory, communication is a form of action and is itself driven by certain goals: utterances can and should be analysed in terms of the goals that they are
supposed to accomplish, see, for instance [Bunt, 2000] [Beun, 1994]. If this is accepted, it is obvious that a system that has no means to represent the goals of the participants cannot fully grasp the communicative function of an utterance. As a consequence, its ability to engage in meaningful dialogue is necessarily rather shallow. To mend this, one needs a more comprehensive model of an agent that also incorporates a formal treatment of intentions.

7.5 Conclusions

In this chapter we have considered the question how agents (with private knowledge states formalised in type theory) might communicate, despite the privateness of their information.

Given a situation where two agents have knowledge states formalised in type theory, we have constructed a simple framework to study information exchange between the two agents. The framework is based on:

- A simple coding/decoding scheme that allows agents with a shared vocabulary to construct messages that are meaningful to both agents.

- An extension of the knowledge states of the agents that allows them to distinguish private and shared information.

- The addition of pragmatic 'tags' to messages that allow the agents to indicate how messages are to be processed.

We have shown how the framework allows agents to exchange public messages that are meaningful to both participants, how the receiver can interpret these messages, and how the agents can achieve an exchange of ideas and extend their common knowledge. Note, that though we have defined a notion of common knowledge, the situation that we consider is a realistic one in the sense that all information is really distributed over the participants. As information is really distributed, the 'general' nature of the common vocabulary can now be shown to constrain the messages that the agents can exchange. Formally these constraints manifest themselves as binding problems for variables that cannot be eliminated through the use of the common vocabulary. As we have seen, these limitations will often prevent agents from referring directly to objects, even if these objects are commonly known.

To communicate the identity of objects in spite of the (realistic) limitations of their vocabulary, the agents can use messages with 'negative' interpretations, in which extending segments of contexts are regarded as 'specifications', i.e. as requirements that need to be fulfilled. Formally, the possibility to interpret segments 'negatively' rests on the ability of the agents to construct realisations for extending segments of their contexts. As our examples show, both the 'negative' and the 'positive' interpretation of types
have natural uses in a communicative setting, and play a crucial role in the exchange of information.

Summarising, the type theoretical modelling of knowledge states provides a simple and practical framework in which knowledge exchange can be studied from a computational viewpoint, and may even be implemented.

The framework may well be applicable in the implementation of intelligent user interfaces that can communicate with users and adapt their messages to the actual knowledge of the user. To realise this, we have to be able to translate the ‘annotated segments’ into a public language that is acceptable to human users.

Particularly interesting candidates for such a language are natural languages, like English. In this respect, it is significant that this framework, using type theoretical knowledge states as a semantic background of natural language utterances, is surprisingly compatible with existing linguistic theories. In fact, various linguistic phenomena have already been formulated within the framework: the resolution of definite descriptions (including anaphora and uses of deixis [Beun & Kievit, 1996], question/answer relations [Piwek, 1998], scope ambiguities [Alin & al., 1995], and, as we have seen in chapter 6, the treatment of presuppositions and other discourse relations. Given that natural language sentences can be generated by ‘sugaring’ type theoretical expressions, as shown in [Mäenpää & Ranta, 1990], it may well be that a further extension of the framework, and in particular the modelling of intentions, opens up possibilities to construct intelligent user-friendly interfaces that have limited natural language understanding capabilities.
Chapter 8

Conclusions

8.1 Discussion

The main goal of this dissertation has been to construct an integrated and
computational model of the knowledge state of an agent in which reason,
observation and communication are integrated.

The first question that we investigated is how to relate the knowledge
of an agent to an external reality. As we had to model this relation in a
computational way, we were forced to model it from a subjective viewpoint,
relating an agent’s knowledge not to the world as it is, but to the world as
it appears to him. To model the subjective knowledge of the agent we used
type theory. Type theory is a formalism in which all asserted propositions
have explicit justifications. It is rooted in the constructive tradition where
emphasis is not on abstract truth, but on provability. The meaning of a
proposition lies not in its truth conditions, but in the ability to recognise (or
construct) a justification for this proposition.

In chapter 3, we have taken this view outside the mathematical sphere\(^1\)
and applied it to the understanding of an external world, to achieve a
computational modelling of Fregean sense. This has led to a picture where agents
understand the world in terms of their own concepts. These concepts acquire a (entirely private) meaning which depends on the ability of an agent
to recognise observed instances of these concepts. As we represented the
agent’s knowledge in type theory, the concepts of the agent are mapped to

\(^1\)A suggestion to this effect is found in [Dummett, 1975, pp. 108-109]:

... this meant replacing the notion of truth by that of proof: evidently, the
appropriate generalisation for this, for statements of an arbitrary kind, would
be the replacement of the notion of truth, as the central notion of the theory
of meaning, by that of verification; to know the meaning of a statement is,
on such a view, to be capable of recognising whatever counts as verifying
the statement, i.e. as conclusively establishing it as true ...
types. In accordance with the constructive tradition, the meaning of these types lies in the agent's ability to construct inhabitants of these types. This ability relates the knowledge of the agent to the external world: the agent can recognise instances of some of these concepts, i.e. inhabitants of certain types in the external world.

The resulting relationship between knowledge and reality is computational and subjective, and can be computed by the agent himself. This allows the agent to extend his knowledge through observation, and to apply this knowledge in situations that he has not encountered before. A less desirable but entirely realistic consequence of this approach is that the agent can make mistakes in the interpretation of the external world.

The resulting knowledge states are intensional; they allow an agent to distinguish between coreferent expressions, and explain how concepts like 'unicorn' or 'greatest prime' that do not or even cannot have any instances in the real world, can still be meaningful. Interestingly, the proposed mechanism of semantic grounding reproduces certain familiar practical problems: the agent may have difficulty in distinguishing objects that are similar, and may be confused by ambiguous stimuli.

Note that the meaning of a concept, i.e. the ways in which an agent can recognise an instance of the concept, depends on the observational abilities of this agent. These abilities are explicitly defined in terms of interpretable primitives, and can be changed at will. If this happens, the ways in which an agent recognises the instances of various concepts will also change. Typically, the number of ways to recognise a certain concept will increase or decrease, as observational abilities, like the perception of colour, are added or removed. This allows us to manipulate and study the effects of different kinds of observational 'grounding' for a given agent, enriching or impoverishing his observational abilities.

An additional advantage of the semantically grounded deductive knowledge states proposed here, is that they can be combined with a foundational approach to belief revision.

The second question that we investigated, is how the type-theoretical knowledge states can be applied to reason about an external reality that changes in time. In chapter 4, we developed an ontology and a model of a changing universe. After formalising the model in predicate logic, we showed how one can reason rigorously about the history of a given universe, and prove facts that hold in all possible histories of this universe that are consistent with known facts about this history. The resulting reasoning method is monotonic and not based on exhaustive search. A given body of knowledge can gradually be extended with new facts, without invalidating any earlier results.

As the method developed in chapter 4 uses predicate logic, we have, in chapter 5, translated the formalisation to type theory, and combined it with the modelling of knowledge states proposed earlier. In this way, we have
been able to create a formal framework in which the development through time of a knowledge state of an agent that observes his surroundings can be described. As the knowledge state of the agent is incremental, new facts that are observed can directly be added to the agent’s body of knowledge. With each new fact that is observed, the agent’s knowledge about the history of the universe gradually increases.

In the resulting model there is no intrinsic difference between past, present and future: the apparent openness of the future is explained as an epistemic phenomenon: an agent can observe the present, and retrieve observational information about the past, but he has no direct source of information about the future. Though the agent can reason about the future, he can only deduce what will necessarily happen. Of course, the agent himself also lives in time. His knowledge increases with new observations, transforming an ‘open’ future in a ‘known’ past.

Using his reasoning abilities, the agent can plan the future: by assuming that a desired goal will indeed be reached, he can deduce information about necessary events that lie between the present state and the desired goal. Through this mechanism the agent may also be able to conclude that certain goals are unreachable, without having to go through an exhaustive search.

In chapters 6 and 7, we investigated the question whether the ‘lonely’ observing and reasoning agents that we modelled can communicate with one another.

In particular, in chapter 6, we have shown how type theoretical knowledge states can be related to Discourse Representation Theory (DRT), a well-known theory of natural language interpretation. We have presented a translation from the semantic representations of DRT, so-called DRSs, to type theory, and have shown how DRT’s truth-conditional semantics can be considered as a limiting case of a finer-grained ‘mentalistic’ semantics based on satisfiability with respect to a given context.

Having developed explicit models of knowledge states, we have been able, in chapter 7, to investigate how information transfer between knowledge states might be achieved. Assuming that types can be related to words of a common vocabulary, and that agents have the ability to mark parts of their knowledge as shared, we have identified basic mechanisms that the agents may employ in the exchange of information. Through a number of examples it has been shown how these mechanisms may indeed achieve information transfer between the agents. The examples illustrate how the information that the agents can exchange is dependent upon their common knowledge, and how the common knowledge of the agents is extended when information is transferred.

Finally, we have briefly discussed the relevance of these ideas to the implementation of man-machine interfaces.

Summarising, we have modelled the knowledge of an agent about a dy-
namic external reality in a computational way, emphasising in particular the role of reasoning, observation and communication processes, and their incremental effects on the agent’s knowledge state.

As our model had to be computational, we were forced to abandon the truth-conditional approach to meaning in favour of more of a more subjectivist and verificationist account centered around ‘justifications’. This approach allowed us to formulate new, computational and compatible answers to a number of well-known questions related to intensionality, time and communication. Due to the computational nature of these answers, they may be relevant to the realisation of intelligent software agents that can assist humans in their dealings with cyberspace.

8.2 Future research

The questions that we have considered leave many important issues unaddressed that are important for the development of a computational and integrated model of a knowing agent. In this section we will briefly indicate how some of these issues may be addressed in ways that are compatible with the approach that we have taken in this thesis.

8.2.1 Reflexivity

One obvious and interesting direction of future research is to investigate how we might enable the agent to reason about his own knowledge state. One possibility is to endow the agent with the ability to consider his own utterances as objects.

It may be possible to construct an agent that reasons about a universe that contains written English sentences as objects. If the agent has some language-understanding capabilities, and is able to answer questions about this universe, one may then point at some language construct S and ask: “Did you say this?” Or even: “Do you believe this?” The point is that the agent does not have to interpret such an utterance as referring to the literal string of characters that makes up the sentence S. The agent can also interpret this utterance as referring to the meaning of S, for which he has a formal representation. The implications of such a move are very far-reaching. An important question is to which extent one could give a satisfying analysis of propositional attitudes, using such a ‘quotational’ approach.

Much has already been said about similar ideas in the literature. For instance Cresswell (1988) gives a detailed and interesting analysis of one particular version of this idea [Cresswell, 1988]. Kripke’s famous ‘puzzle about belief’ involving Pierre and his adventures in London, [Kripke, 1979] deals with similar ideas. In all these cases, the conclusion is, that it is not possible
to understand assertions and beliefs as a direct relation between an agent and an utterance. It may seem, therefore, that such a treatment to propositional attitudes cannot work.

But this impression is quite mistaken. For all such criticisms are directed at one particular version of this approach, a rather trivial version in which the whole idea of meaning is avoided. Cresswell calls this version ‘inscriptionsalism’. He shows that one cannot trivially analyse a sentence like ‘Alfred believes that cows eat grass’ as a relation between Alfred and the uninterpreted string “Cows eat grass”. The problem is that such a trivial analysis is not able to explain how such examples behave under various translations. The same is also true for a sentence involving direct speech, like “Alfred says that cows eat grass”.

But these criticisms do not apply to an approach in which one analyses such sentences as a relation between a person and the meaning of a given string. Cresswell is quite explicit about this; for instance, in [Cresswell, 1988] pp. 108, he discusses an approach to indirect speech that he does not criticise:

... and so (in this viable approach) the meaning of ‘SAY’ relates a person to a sentence and its meaning. I accept that such a theory might be viable, for I myself accept a theory in which the meaning of ‘SAY’ relates a person to its meaning. But for that reason I would not describe such a theory as a version of inscriptionsalism, if by this is meant a theory which avoids commitments to meanings ...

If Cresswell is right, it may indeed be possible to get a valid account of indirect speech or propositional attitudes in the presence of a mechanism that allows one to quote sentences in order to refer to the ‘meaning’ of these sentences. This is an option that seems attractive. When we ask an agent “Do you believe this” (pointing at some sentence) the agent must, in order to answer this question, first analyse the string pointed at (in the same way, in principle, as he analyses every other utterance). After parsing and interpreting it, he can decide on the matter by finding out whether the resulting annotated extending segment (which is of course the agent’s private representation of the meaning of the utterance) does have a realisation in his knowledge state. So it seems that such an account of a quotational theory of propositional attitudes, which actually ‘unquotes’ the sentence to get at its meaning, is impervious to this well-known criticism, and may be worth a try.

8.2.2 Concept learning

The agents whose knowledge states we have modelled thus far, have one very strong and rather unfortunate limitation. They can ‘learn’ new facts, but
they cannot learn new concepts. This makes them rather rigid, and does not present us with a method to develop ‘evolving’ agents, that do not have to be programmed, and are able to adapt themselves to new developments in their environment.

We do not want to go into the subject of concept learning itself. What interests us here, is whether (and how) a ‘concept learner’ if one were provided could be integrated with the the type-theoretical knowledge states that we have been concerned with. Imagine that we somehow have a recogniser (based on neural networks, say) that is able to generate new useful concepts from observational data. Typically, the network tunes in to a certain concept spontaneously, and when it is stimulated often enough, it comes up with a new concept, like ‘sponge’.

It is quite easy to extend the knowledge state of a given agent to accommodate the new concept. A new type ‘sponge’ is declared, and the symbol ‘sponge’ is taken as a member of the set $\mathcal{S}$ of interpretable primitives. Now the agent is able to recognise sponges.

However, the agent must fit the new concept in his given conceptual scheme, and also be able to formulate and collect additional knowledge about sponges. To fit the new concept in the agents given conceptual scheme, it is essential that the agent is able to detect that the new concept is a refinement of other, already existing concepts. For instance, ‘sponge’ would probably be a subtype of ‘thing’, and may or may not be related to such other concepts as ‘animal’, ‘plant’ or ‘living being’. How these subtype relations may be detected is a subtle question that may also depend on the construction of the ‘concept learner’ that we assumed to be available, and it is a question that we cannot answer here without further research.
Bibliography


BIBLIOGRAPHY


Appendix A

A robot book

In this appendix one finds a ‘book’ in type theory, containing a proof that is similar to the first example proof in chapter 4. The book can be seen as consisting of five parts:

- The first part contains some logical preliminaries, and introduces moments, events, and the function \( \downarrow \) from time slices to time slices.

- The second part contains the translation in type theory of axiom 4.11, as discussed in subsection 5.2.4.

- The third part contains general information about the robot world. (To save space, we have left out those things that are not needed for the proof in question.)

- The fourth part consists of data that are derived from the observations. Robot \( r_1 \) is declared to be inside a room at some moment \( t_1 \), and outside this room at a later moment \( t_2 \).

- The fifth part consists of a number of definitions that are used to prove various simple results:

  - There is an event in which the robot ‘leaves’ the room.
  - This event is shown to be a ‘move’ event.
  - The position just after the move is adjacent to the position just before the move.
  - Therefore, the robot is at the border of the room just before the move.

This last definition is in fact a proof of (the type-theoretical equivalent of) equation 4.20.

The type system uses two sorts, ( \( * \) and \( @ \)), and two axioms ‘\( * : \Box \)’ and ‘\( @ : \Box \)’. The sorts are used to distinguish between time slices — that
are inhabitants of inhabitants of $@$ — and 'normal' mathematical objects — that are inhabitants of inhabitants of '*'. This allows a polymorphic treatment of a function like ‘\[’ which in the text below is introduced as:

\[ \text{jump : } (\Pi' \rightarrow (\Pi X : T. (\Pi M : \text{moment}, T))) \].

Notation: obviously, 'lambda' means: "\[", 'pi' means '\Pi', the symbol '<' stands for inheritance, and '=' is used for definitions. Also, as in Prolog, variables are indicated by capitals, and the direction of clauses is inverted.

## A.1 Preliminaries

We first introduce some logical preliminaries, moments and events, the functions 'b' (be) and 'a' (ae), the ordering '<' (prec) between moments and events, and the function '\[' (jump) from time-slices and moments to time-slices.

```prolog
thing : @,
time : *,

eq =
lambda( T : *,
lambda( X : T,
lambda( Y : T,
pi( G : pi(P:T,*),
pi( F : (G-X), (G-Y)))))) :
pi(T : *, pi(X : T, pi(Y : T,*))),

bot = pi( P : * , P)
: *,

not = lambda(P : *, pi(H : P , bot))
: pi( Q : *,*),

event < time : *

moment < time : *

prec : time => time => *,
```
A.2. AXIOMS ABOUT TIME

ae : event  => moment,
be : event  => moment,

ax1-E   : prec-(be-E)-E  <= [ E : event ],
ax2-E   : prec-E-(ae-E)  <= [ E : event ],
jump-S-R-T : S  <=
             [ S : @ , R : S,  T : moment],

A.2 Axioms about time

The following axioms form the translation of axiom 4.11 in the type-theoretical setting, as discussed in subsection 5.2.4.

ear0-TS-R-T1-T2-P-Q-Z-Q2 : event  <=
                             [ TS : @,  R : TS,  T1 : moment,  T2 : moment ,
                             P : pi( Xr : TS, *),
                             Q : (P-(jump-TS-R-T1)) ,
                             Z : prec-T1-T2 ,
                             Q2 : not-(P-(jump-TS-R-T2)) ],

ear1-TS-R-T1-T2-P-Q-Z-Q2 :
prec-T1-(ear0-TS-R-T1-T2-P-Q-Z-Q2) <=
                             [ TS : @,  R : TS,  T1 : moment ,
                             T2 : moment ,
                             P : pi( Y : TS, *),
                             Q : (P-(jump-TS-R-T1)) ,
                             Z : prec-T1-T2 ,
                             Q2 : not-(P-(jump-TS-R-T2)) ],

ear2-TS-R-T1-T2-P-Q-Z-Q2 :
     (prec-(ear0-TS-R-T1-T2-P-Q-Z-Q2)-T2) <=
                             [ TS : @,  R : TS,  T1 : moment,
                             T2 : moment ,
                             P : pi( Y : TS, *),
                             Q : (P-(jump-TS-R-T1)) ,
                             Z : prec-T1-T2 ,
                             Q2 : not-(P-(jump-TS-R-T2)) ],
ear3-TS-R-T1-T2-P-Q-Z-Q2 :
(P-(jump-TS-R-(be-(ear0-TS-R-T1-T2-P-Q-Z-Q2))))

<=
[ TS : Ø, R : TS , T1 : moment ,
  T2 : moment ,
P : pi( Y : TS, *),
Q : (P-(jump-TS-R-T1)) ,
Z : prec-T1-T2 ,
Q2 : not-(P-(jump-TS-R-T2)) ],

ear4-TS-R-T1-T2-P-Q-Z-Q2 :
not-(P-(jump-TS-R-(ae-(ear0-TS-R-T1-T2-P-Q-Z-Q2))))

<=
[ TS : Ø, R : TS , T1 : moment , T2 : moment ,
P : pi( Y : TS, *),
Q : (P-(jump-TS-R-T1)) , Z : prec-T1-T2 ,
Q2 : not-(P-(jump-TS-R-T2)) ],

ear5-TS-R-T1-T2-P-Q-Z-Q2-T-G-H :
(P-(jump-TS-R-T))

<=
[ TS : Ø, R : TS , T1 : moment ,
  T2 : moment ,
P : pi( Y : TS, *),
Q : (P-(jump-TS-R-T1)) ,
Z : prec-T1-T2 ,
Q2 : not-(P-(jump-TS-R-T2)) ,
T : moment ,
G : prec-T1-T ,
H : prec-T-(ear0-TS-R-T1-T2-P-Q-Z-Q2) ],

A.3 Concepts and invariants in the robot world

Next, some concepts and facts about the robot world are introduced. We start with the introduction of time slices of robots and tiles. We declare positions, colours, some knowledge about adjacency of positions, and the notion of a border. We also introduce some knowledge about events. The 'movax' axiom, which corresponds to equation 4.14, has been simplified, to minimize reasoning about equality.
robot \textless{} thing : 0, \\
tile \textless{} thing : 0, \\
position : *, \\
room : position \Rightarrow{} * , \\
colour : *, \\
move-R-E : * \iff [ R : robot , E : event ], \\
col : robot \Rightarrow{} colour, \\
pos : robot \Rightarrow{} position, \\
movax-R-E-P-Q-W : move-R-E \iff \\
[ R : robot , E : event , P : \Pi( X : position, * ) , \\
Q : P-(pos-(jump-robot-R-(be-E))), \\
W : not-(P-(pos-(jump-robot-R-(ae-E)))) ], \\
adj : position \Rightarrow{} position \Rightarrow{} * , \\
axmov1-R-E-M : \\
adj-(pos-(jump-robot-R-(be-E))) \\
-(pos-(jump-robot-R-(ae-E))) \iff \\
[ R : robot , E : event , M : move-R-E ], \\
border : position \Rightarrow{} * , \\
bor-P-Q-Z-H-G : (border-P) \iff \\
[ P : position , Q : position , Z : (adj-P-Q), \\
H : (room-P) , G : not-(room-Q) ], \\

A.4 Observational data

This section of the book introduces the data that follow from observations that we discussed in section 5.3.

r1 : robot,
% assume there is a robot

tt1 : moment,

tt2 : moment,
% there are two moments

q1 : (prec-\(tt1-\)\(tt2\)),
% \(tt1\) precedes \(tt2\)

q2 : room-(pos-(jump-robot-\(r1-\)tt1)),
% at \(tt1\) the robot is in the room.
q3 : not-(room-(pos-(jump-robot-\(r1-\)tt2))),
% at \(tt2\) the robot is no longer in the room.

inside = lambda( R : robot , (room-(pos-R))) :
    pi( R : robot , *),
% define the predicate ‘inside’.

A.5 Conclusions

Finally, the definitions that show respectively, that:

1. There is a first event that changes the truthvalue of ‘inside’ for the robot ‘\(r1\)’.

2. This event is a move event.

3. Positions of the robot just before and after the event must be adjacent.

4. The robot was on the border of the room just before the event in question.

\(eq119 =
\)

(ear0-robot-\(r1-\)tt1-\(tt2\)-inside-q2-q1-q3 ) : event,
% ‘eq119’ is the first event between tt1 and tt2 that changes
% the predicate ‘inside’ for the robot \(r1\).

inroom =
(ear3-robot-\(r1-\)tt1-\(tt2\)-inside-q2-q1-q3)
inside-(jump-robot-r1-(be-eq119)),
% 'inroom' is a proof that robot r1 is inside just before
% this event

outroom =
(sear4-robot-r1-tt1-tt2-inside-q2-q1-q3)
: not-(inside-(jump-robot-r1-(ae-eq119))),
% 'outroom' is a proof that robot r1 is outside just after
% this event

fact123 = (movax-r1-eq119-room-inroom-outroom) : move-r1-eq119,
% 'fact123' proves that this event was a 'move' event.

fact124 =
(axmov1-r1-eq119-fact123) : adj-(pos-(jump-robot-r1-(be-eq119)))
-POS-(jump-robot-r1-(ae-eq119))),
% 'fact124' proves that positions just
% before and after event 'eq119' are adjacent.

fact125 =
(bor-(pos-(jump-robot-r1-(be-eq119)))
-POS-(jump-robot-r1-(ae-eq119))
-fact124-inroom-outroom)
: border-(pos-(jump-robot-r1-(be-eq119)))

% 'fact125' is the proof that r1 is at the border
% just before this event.
% This was the desired result.

}).
Samenvatting

De centrale vraagstelling van dit proefschrift is hoe verschillende vormen van intelligent gedrag, zoals communicatie, inferentie, maar ook observatie, in hun onderlinge samenhang gemodelleerd kunnen worden rond een computatieneel model van kennis.

We nemen hierbij aan dat de bedoelde kennis geregerepresenteerd kan worden d.m.v. een veranderlijke structuur, een kennistoestand. Psychologisch realisme wordt door ons niet nagestreefd, we streven slechts naar een berekenbaar model van informatie: we eisen dat alle processen die plaatsvinden bij het raadplegen of veranderen van deze kennis (b.v. op grond van waarnemingen of van communicatie) berusten op berekeningen die kunnen worden uitgevoerd door het kennisende subject zelf. De motivatie achter deze eis is tweeledig: enerzijds is het een nodige voorwaarde om zo een model te kunnen implementeren op een computer, anderzijds is het een nuttig methodologisch uitgangspunt dat voorkomt dat we een theoretische model formuleren dat in werkelijkheid niet kan bestaan.

Dat we onszelf op deze wijze dwingen alleen uit te gaan van het persoonlijke gezichtspunt van het kennisende subject, heeft ingrijpende consequenties voor de wijze waarop de relatie tussen kennis en werkelijkheid kan worden gemodelleerd. Deze relatie dient immers geheel te berusten op de concepten die het kennisende subject zelf gebruikt om deze werkelijkheid te systematiseren en te begrijpen. Mathematisch baseren wij ons model daarom niet op de verzamelingentheorie, die extensioneel is, en uitgaat van (voor een kennisende subject) onberekenbare noties zoals de gelijkheid van objecten, maar op type-theorie, een formalisme dat voortkomt uit de z.g. constructieve traditie in de wiskunde en dat uitgaat van een berekenbaar begrip van herkenning van ‘bewoners’ van verschillende soorten, de z.g. types.

Nadat in hoofdstuk 2 een inleiding in type-theorie is gegeven, wordt in de latere hoofdstukken aangegeven hoe we met behulp hiervan een model van een kennistoestand kunnen opbouwen. Dit leidt tot een constructie waarbij de kennis van het subject over de werkelijkheid in type-theorie wordt geregerepresenteerd, en de koppeling tussen kennis en werkelijkheid gelegd wordt door dat van een aantal types wordt aangenomen dat ze waarneembaar zijn, d.w.z. dat het subject in staat is ‘bewoners’ van deze types in de buitenwereld te herkennen. Op deze wijze verkrijgen we een berekenbare koppeling tussen kennis en de werkelijkheid, waarbij o.a. duidelijk is hoe het subject zijn kennis kan uitbreiden en toepassen op grond van observaties. De kennis van het subject binnen dit model geeft nergens weer wat de objecten werkelijk zijn, maar weerspiegelt alleen hoe ze gekend worden.

In hoofdstuk 3 wordt aangetoond dat dit leidt tot een berekenbare en fijnmazige notie van betekenis die in staat is een bevredigend antwoord te verschaffen op een aantal basisvragen uit de semantiek. Interessant is verder
dat de genoemde constructie zeer realistische eigenschappen vertoont: het is b.v. mogelijk om te modelleren hoe er bij het subject verwarring kan optreden.

In de hoofdstukken 4 en 5 is de vraag onderzocht hoe de kennis van het subject zo kan worden georganiseerd, dat deze bovendien in staat is te redeneren over veranderingen in de werkelijkheid, en over de effecten van gebeurtenissen. Het blijkt mogelijk om een bestaand veranderlijk domein zodanig te beschrijven in termen van gestructureerde objecten, dat de effecten van gebeurtenissen in deze objecten kunnen worden gelocaliseerd, en dat men gegevens over de toestand van het domein op verschillende tijdstippen binnen een monotoon naamwerk kan combineren. Een kennisopstelling kan hierdoor in de loop van de tijd worden uitgebreid met nieuwe gegevens die vrijelijk kunnen worden gecombineerd zonder dat er inconsistenties ontstaan. Op grond hiervan kan het subject redeneren over het verleden en de toekomst en kan het de noodzakelijke consequenties van toekomstige doelen voorzien.

In de hoofdstukken 6 en 7 gaan we in op de vraag hoe twee subjecten met verschillende kennisopstellingen onderling kunnen communiceren, ondanks het subjectieve karakter van hun kennis. Eerst wordt aangegeven hoe de door ons geconstrueerde kennisopstellingen kunnen worden gerelateerd aan Discourse Representatie Theorie (DRT), een bekende theorie van natuurlijke taal interpretatie. We laten zien hoe de semantische structuren van DRT kunnen worden vertaald naar type-theorie, en dat de waarde-conditionele semantiek van DRT kan worden beschouwd als een limietgeval van de meer fijnmazige semantiek van type-theoretische kennisopstellingen.

Uitgaand van de expliciete representatie van kennisopstellingen zijn we vervolgens in staat, in hoofdstuk 7, om te modelleren hoe informatieoverdracht tussen kennisopstellingen in principe mogelijk is. We identificeren een aantal basismechanismen die hierbij een rol spelen en geven enige voorbeelden. Deze voorbeelden illustreren hoe de informatie die kan worden overgedragen afhankt van reeds bestaande gemeenschappelijke informatie en hoe de gemeenschappelijke informatie toeneemt door kennisoverdracht.
Curriculum Vitae


Huidig adres:
IPO, Center for User-System Interaction
Den Dolech 2, PO Box 513, 5600 MB Eindhoven, Nederland
e-mail: R.M.C.Ahn@tue.nl
Titles in the IPA Dissertation Series


P.M. Achten. Interactive Functional Programs: Models, Methods, and Implementation. Faculty of Mathematics and Computer Science, KUN. 1996-3

M.G.A. Verhoeven. Parallel Local Search. Faculty of Mathematics and Computing Science, TUE. 1996-4

M.H.G.K. Kesseler. The Implementation of Functional Languages on Parallel Machines with Distributed Memory. Faculty of Mathematics and Computer Science, KUN. 1996-5


H. Doornbos. Reactivity Arguments and Program Construction. Faculty of Mathematics and Computing Science, TUE. 1996-8

D. Turi. Functorial Operational Semantics and its Denotational Dual. Faculty of Mathematics and Computer Science, VUA. 1996-9


N.W.A. Arends. A Systems Engineering Specification Formalism. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. Normalisation in Lambda Calculus and its Relation to Type Inference. Faculty of Mathematics and Computer Science, TUE. 1996-12


M.M. Bonsangue. Topological Dualities in Semantics. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. Algorithms for Graphs of Small Treewidth. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. Process-algebraic Transformations in Context. Faculty of Computer Science, UT. 1997-02


C.J. Bloo. Preservation of Termination for Explicit Substitution. Faculty of Mathematics and Computing Science, TUE. 1997-05


F.A.M. van den Beuken. A Functional Approach to Syntax and Typing. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. Ins and Outs in Refusal Testing. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. A Discrete-Event Simulator for Systems Engineering. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. Scheduling with Communication for Multiprocessor Computation. Faculty of Mathematics and Computer Science, UU. 1998-03


E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model.* Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing.* Faculty of Computer Science, UT. 1999-02


C.H.M. van Kemenade. *Recombinative Evolutionary Search.* Faculty of Mathematics and Natural Sciences, Univ. Leiden. 1999-04


J.P. Warners. *Nonlinear approaches to satisfiability problems.* Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods.* Faculty of Computer Science, UT. 1999-09

P.R. D’Argenio. *Algebras and Automata for Timed and Stochastic Systems.* Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems.* Faculty of Mechanical Engineering, TUE. 1999-11


R.S. Venema. *Aspects of an Integrated Neural Prediction System.* Faculty of Mathematics and Natural Sciences, RUG. 1999-13


T.E.J. Vos. *UNITY in Diversity: A stratified approach to the verification of distributed algorithms.* Faculty of Mathematics and Computer Science, UU. 2000-02


J. Fey. *Design of a Fruit Juice Blending and Packaging Plant.* Faculty of Mechanical Engineering, TUE. 2000-06

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events: a Computational Approach to Knowledge, Observation and Communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02