INCREMENTAL CONCEPTUALISATION FOR LANGUAGE PRODUCTION

Markus Guhe
INCREMENTAL CONCEPTUALISATION FOR
LANGUAGE PRODUCTION

Dissertationsschrift zur Erlangung des Grades eines
Doktors der Naturwissenschaften
am Fachbereich Informatik
der Universität Hamburg

vorgelegt von
MARKUS GUHE

aus Papenburg

Oktober 2003
SUMMARY

Conceptualisation is the cognitive task that takes non-linguistic knowledge and generates preverbal messages (semantic structures), which are linguistically encodable. It can, therefore, be seen as a mediator between perception and other cognitive faculties on the one hand and language on the other. Up to now there exists no computational model of the conceptualiser. In this investigation I provide the first one.

Investigating conceptualisation suffers from the difficulty that it is never directly observable but only via another modality, first of all language. To overcome this difficulty I investigate how descriptions of events in an online setting can be generated. This means, firstly, that I consider verbal descriptions of events – mostly motion events –, and, secondly, that I use a setting in which the verbalisations are produced while the events take place. So, the setting is strongly data-driven. The temporal interleaving of processing perceptual input and generating verbal output allows to correlate input and output, and is, therefore, a means to overcome the difficulty. Additionally, the online setting reduces the complexity of conceptualisation to a degree where it is possible to account for the open-ended issue of conceptualisation with a computational model. In the setting I use the conceptualisation task is subdivided into four sub-tasks: construction of a hierarchical conceptual representation, selection of the events that are described verbally, linearisation of the selected events, and generation of preverbal messages describing these events.

The computational model that is described in this investigation is $\text{INC}$, the incremental conceptualiser. As the name already suggests, $\text{INC}$ uses an incremental mode of operation to cope with the dynamics of the online setting, because incremental processing considers only the changes in the input. The characteristic behaviour of incremental models is that they produce output before all input, which may be relevant for the correct and complete computation of the corresponding output, is available. It can be achieved by the parallel (cascaded) processing of a sequential information stream. This means that as soon as a piece of information (an increment) was processed on one stage it is passed on to the next stage. In its strong form I call this property Extended Wundt’s Principle: input is processed and output is generated as soon as it is available.
Based on different kinds of incrementality proposed in the literature I provide a general definition of incrementality and discuss the dimensions along which it can vary. Apart from a cascade of incremental processes an incremental models consist of a shared representation of the model knowledge on which the processes operate. As a blueprint for developing incremental models based on the definition I provide a formalisation in the specification language Z.

Cascaded architectures have a unidirectional information flow with no feedback, which keeps the model efficient and simple. Since this is also a source of errors, I propose a relaxation without sacrificing the unidirectionality of the information flow by allowing indirect feedback. In this kind of feedback no explicit information is given back, but the effects of computations influence previous components in the cascade.

\textsc{inC}'s behaviour is adapted by assigning different values to parameters so that different preverbal messages can be generated for the same input. The input in the simulations that are carried out this way is identical to the one used for verbalisation studies in which participants have to perform the same task. This makes it possible to compare the output of the simulations to the observed verbalisations. In this way the cognitive adequacy of \textsc{inC} is evaluated. The simulations show that \textsc{inC} is a realistic, ie cognitively adequate model of the human conceptualiser.


Das Modell, das in dieser Arbeit beschrieben wird, ist INC (incremental conceptualiser). Wie der Name bereits besagt, verwendet INC einen inkrementellen Verarbeitungsmodus, um die Dynamik der online Bedingung zu bewältigen; denn inkrementelle Verarbeitung betrachtet nur jeweils die Änderungen in den Eingabedaten. Das charakteristische Verhalten eines inkrementellen Modells ist, daß es Ausgaben produziert, bevor alle Eingaben, die für die korrekte und vollständige Berechnung der Ausgabe relevant sein können, vorhanden sind. Dies wird üblicherweise durch die parallele (kaskadierte) Verarbeitung eines sequentiellen Informationsstroms erreicht. Das heißt, sobald ein Informationsstück (ein Inkrement) auf einer Ebene verarbeitet wurde, wird es an die nächste Ebene weitergegeben. In seiner starken
Ausprägung nenne ich dies das Erweiterte Wundt'sche Prinzip: Eingaben werden verarbeitet und Ausgaben werden produziert, sobald sie verfügbar sind.


Kaskadierte Architekturen haben einen unidirektionalen Informationsfluß ohne Feedback, was die Modelle effizient und einfach hält. Da dies allerdings auch eine Fehlerquelle ist, schlage ich eine Aufweichung dieses Prinzips durch Zulassen indirekten Feedbacks vor, ohne damit die Unidirektionalität des Informationsflusses aufzugeben. Bei dieser Art von Feedback werden keine Informationen explizit zurückgegeben, aber gewisse Effekte von Berechnungen beeinflussen vorhergehende Komponenten der Kaskade.

\textit{INCS} Verhalten wird über Parametersetzungen beeinflußt, so daß verschiedene präverbale Botschaften für dieselbe Eingabe erzeugt werden können. Die Eingaben in diesen Simulationen sind identisch zu denen, die in Verbalisierungsstudien verwendet werden. Auf diese Weise können die Simulationsergebnisse mit den beobachteten Verbalisierungen verglichen und die kognitive Adäquatheit von \textit{INCS} ermittelt werden. Die Simulationen zeigen, daß \textit{INCS} ein realistisches, d.h. kognitiv adäquates Modell des menschlichen Konzeptualisiersers ist.
# CONTENTS

| List of terms | xiii |
| List of figures | xvi |
| Preface | xvii |

## 1 Introduction

1.1 Thinking while speaking ........................................ 1
1.2 Incremental conceptualisation: an example .................. 3
1.3 Research questions ............................................. 10
1.4 Questions that are only treated in passing .................. 12
1.5 Disciplines and methods ....................................... 14
1.6 Effects of incrementality on representations ................. 18
1.7 Outline ................................................................ 21

## 2 Language production

2.1 Investigating language production .............................. 26
2.2 Conceptualisation as mediator for language ................ 27
2.3 Levelt’s tripartite architecture ................................. 29
2.4 Modularity of conceptualisation ................................ 34
2.5 Non-modular aspects: conceptualisation and other cognitive tasks 36
2.6 Modular aspects: language specificity of conceptualisation 39

## 3 Conceptualisation

3.1 Investigating conceptualisation ................................ 43
3.2 Conceptualisation tasks .......................................... 46
3.3 Conceptualisation and perception .............................. 53
3.4 Incremental conceptualisation: an outlook .................. 54
## CONTENTS

### 4 Conceptual representations

4.1 Concepts ........................................... 57
4.2 A remark on the symbol grounding problem ............... 61
4.3 Referential nets ..................................... 63
4.4 Representations of the external world ..................... 67
4.5 Event representations ................................ 68
4.6 Representations for speaking ........................... 75

### 8 Incrementality

#### 5 Incrementality

5.1 Approaching the phenomenon .......................... 80
5.2 Kinds of incrementality ................................ 83
5.3 A blueprint of incrementality .......................... 89
5.4 Dimensions of incrementality .......................... 100
5.5 Incremental representations ............................ 109

### 6 Formalisation

6.1 Incremental processing ................................ 117
6.2 Referential nets ...................................... 128

### 7 Resources and architecture

7.1 Terminology ........................................... 139
7.2 Bounded rationality ................................... 141
7.3 Anytime processing .................................... 142
7.4 A remark on the frame problem ........................ 145
7.5 Unified cognitive architectures ........................ 146
7.6 Differences to nC ...................................... 149
7.7 Principles used in building nC ......................... 151

### C nC – THE INCREMENTAL CONCEPTUALISER

#### 8 Overall architecture

8.1 Processes ............................................. 155
8.2 Representations ....................................... 157
8.3 Increments ............................................ 158
8.4 Parameters ............................................ 159
8.5 Pre-processing unit (PPU) and perceived entities (Pes) .... 160

#### 9 Current conceptual representation (CCR)

9.1 The representation ..................................... 163
9.2 Traverse buffer ........................................ 166
9.3 Traverse .............................................. 167
CONTENTS

Bibliography 273
Name Index 287
Subject Index 291
# LIST OF TERMS

1. Model .................................................. 14
2. System .................................................. 14
3. Conceptualisation for language production ................. 46
4. Category ............................................... 59
5. Concept ............................................... 59
6. Wundt’s Principle .................................... 82
7. Extended Wundt’s Principle ......................... 82
8. Incrementality ........................................ 90
9. Increment ............................................. 90
10. Update increment .................................. 91
11. Increment stream .................................. 91
12. Incremental process ................................ 91
13. Model knowledge .................................. 92
14. Available knowledge ................................ 93
15. Focussed element .................................. 93
16. Local context ....................................... 93
17. Incremental algorithm ................................ 95
18. Increment buffer ..................................... 96
19. Cascade ............................................... 98
20. Increment model .................................... 98
21. Incremental behaviour ................................ 99
22. Indirect feedback ................................... 104
23. Resource ............................................ 140
24. Allocation ............................................ 140
25. Metric ................................................. 140
26. Adaptation ........................................... 141
27. Traverse ............................................... 165
28. Traverse buffer ..................................... 165
29. Event thread ......................................... 189
30. Incremental preverbal message .................... 191
31. Current preverbal message ........................ 193
32. Monitored traverse .................................. 227
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Example scene: docking of a plane</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>A part-of hierarchy and a subsumption hierarchy</td>
<td>7</td>
</tr>
<tr>
<td>1.3</td>
<td>Example scene: representation of the situation structure</td>
<td>7</td>
</tr>
<tr>
<td>1.4</td>
<td>Example scene: more elaborated representation of the hierarchical event</td>
<td>10</td>
</tr>
<tr>
<td>2.1</td>
<td>Representational systems involved in thought and language</td>
<td>29</td>
</tr>
<tr>
<td>2.2</td>
<td>Level’s blueprint for the speaker</td>
<td>30</td>
</tr>
<tr>
<td>3.1</td>
<td>Overall setting of verbalisation studies and simulating conceptualisation</td>
<td>45</td>
</tr>
<tr>
<td>4.1</td>
<td>The part of the sort hierarchy (lattice) used in the motion event domain</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>Referential net for the example of the docking plane</td>
<td>71</td>
</tr>
<tr>
<td>5.1</td>
<td>A piece of information running through a cascade of processes</td>
<td>81</td>
</tr>
<tr>
<td>5.2</td>
<td>Kinds of incrementality according to Finkler</td>
<td>86</td>
</tr>
<tr>
<td>5.3</td>
<td>A blueprint of incrementality</td>
<td>90</td>
</tr>
<tr>
<td>5.4</td>
<td>Model knowledge, available knowledge, local context, and focussed element</td>
<td>92</td>
</tr>
<tr>
<td>5.5</td>
<td>The CLILS representation for Mary sleeps</td>
<td>113</td>
</tr>
<tr>
<td>5.6</td>
<td>The parallelism constraint for the elliptical sentence (4)</td>
<td>114</td>
</tr>
<tr>
<td>5.7</td>
<td>The underspecified preverbal message of the self-correction in (4)</td>
<td>115</td>
</tr>
<tr>
<td>5.8</td>
<td>A lambda structure that satisfies the constraint graph in figure 5.4</td>
<td>116</td>
</tr>
<tr>
<td>8.1</td>
<td>The architecture of inC</td>
<td>156</td>
</tr>
<tr>
<td>9.1</td>
<td>The CCR including traverse and traverse buffer</td>
<td>165</td>
</tr>
<tr>
<td>9.2</td>
<td>Operations on the traverse buffer</td>
<td>166</td>
</tr>
<tr>
<td>10.1</td>
<td>The construction process</td>
<td>172</td>
</tr>
<tr>
<td>10.2</td>
<td>Construction of an event structure</td>
<td>174</td>
</tr>
<tr>
<td>10.3</td>
<td>Construction’s heuristic for establishing the candidate list</td>
<td>176</td>
</tr>
<tr>
<td>10.4</td>
<td>A rule of the CS</td>
<td>178</td>
</tr>
<tr>
<td>10.5</td>
<td>The concept matcher process</td>
<td>179</td>
</tr>
<tr>
<td>10.6</td>
<td>The function MatchCandidateList</td>
<td>180</td>
</tr>
<tr>
<td>11.1</td>
<td>The selection process</td>
<td>186</td>
</tr>
<tr>
<td>11.2</td>
<td>The standard selection algorithm</td>
<td>186</td>
</tr>
<tr>
<td>11.3</td>
<td>An alternative selection algorithm</td>
<td>187</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>13.1</td>
<td>The outer algorithm of pvm-generation</td>
<td>196</td>
</tr>
<tr>
<td>13.2</td>
<td>The procedure Verbalise</td>
<td>197</td>
</tr>
<tr>
<td>13.3</td>
<td>The function SelectDesignations</td>
<td>199</td>
</tr>
<tr>
<td>13.4</td>
<td>Scene with two moving planes</td>
<td>201</td>
</tr>
<tr>
<td>13.5</td>
<td>Event structure of scene with two moving planes</td>
<td>202</td>
</tr>
<tr>
<td>13.6</td>
<td>The constraint net for the on-going events</td>
<td>203</td>
</tr>
<tr>
<td>14.1</td>
<td>The INC simulator</td>
<td>208</td>
</tr>
<tr>
<td>14.2</td>
<td>A plane turning left</td>
<td>209</td>
</tr>
<tr>
<td>14.3</td>
<td>Final state of the ccr for the plane turning left</td>
<td>210</td>
</tr>
<tr>
<td>14.4</td>
<td>An incremental preverbal message generated by INC for $\text{DOAT} = 0.5$</td>
<td>217</td>
</tr>
<tr>
<td>14.5</td>
<td>Continuation for $\text{AT} = 0.5$</td>
<td>218</td>
</tr>
<tr>
<td>14.6</td>
<td>The second incremental preverbal message for $\text{AT} = 0.4$ and the standard selection strategy</td>
<td>219</td>
</tr>
<tr>
<td>14.7</td>
<td>A simple sketch</td>
<td>220</td>
</tr>
<tr>
<td>14.8</td>
<td>Different sequences of drawing the lines of the crossing</td>
<td>221</td>
</tr>
<tr>
<td>14.9</td>
<td>The event structure of the crossing</td>
<td>221</td>
</tr>
<tr>
<td>14.10</td>
<td>Log-output for the verbalisation of the crossing</td>
<td>222</td>
</tr>
<tr>
<td>15.1</td>
<td>INC extended by a monitor</td>
<td>226</td>
</tr>
<tr>
<td>15.2</td>
<td>The monitor process</td>
<td>228</td>
</tr>
</tbody>
</table>
When we humans talk, a lot of things happen simultaneously: we move our tongue, we listen to what we say, we think about what to say next, we watch the person we are talking to, we feel an itch and scratch, we notice that it is lunch time and we are hungry ... In other words, we are occupied with a plethora of tasks. Nevertheless, we are capable of observing what is happening around us and talking simultaneously at a constant rate, more or less fluently. This is my topic, although I will investigate a somewhat simpler case: the simultaneity of watching a scene and describing it verbally.

In the the last years, when I explained to people what my doctoral thesis is about on scientific conferences, via e-mail, or in informal discussions on various occasions, I got two types of reactions. Some people considered me mad for choosing such an open-ended topic. However, most also showed a considerable interest – even from those deeming me mad. The skepticism was mainly due to the fact that the investigation of conceptualisation lacks from a difficult problem: it is not directly observable but only via other modalities, mainly language. It is also hardly definable what belongs to conceptualisation and what to other areas of cognition. I guess the interest is due to the fact that it is so obvious that thinking and speaking take place simultaneously. But while this is obvious it is also something that can hardly be grasped. This is the point where I think that a computational model (iNC) can provide some new insights. However, it is clear (without five years of thinking) that such a model must be more than incomplete. My hope, therefore, simply is that iNC inspires new thoughts; it certainly is no complete model that explains human thinking.

*  

The translations given in the text are mine. The examples are made up examples until otherwise indicated.

A final remark on the conventions used in the text. Abbreviations are written without full stop. All of them. I mention this, because some people regard this as error, but it is not. In this respect – as in almost all other language related issues
– I stick to the conventions of British English as they are described in the *Oxford Advanced Learner’s Dictionary*.

_I don’t have time; what do I read?_

If you do not wish to read the whole text, I recommend that you start with section 1.2 where the issue of incremental conceptualisation is explained on the basis of an example. Conceptualisation is then described in section 1.2 and in section 4.5 the corresponding representations are discussed. Section 5.2 describes the issue of incrementality in detail, and section 5.3 provides a blueprint of an incremental architecture. The overall architecture of the model inC, in which the ideas of incrementality and conceptualisation are brought together, is laid out in chapter 8 and some simulations I carried out with the implemented system are the topic of chapter 14.

I give a short overview of the main points of most chapters at their beginnings (except for short ones), which can be used for choosing chapters and sections. The introductory chapter contains an overview of the whole text.

_Acknowledgements_

I want to thank my supervisors for discussions and comments regarding this investigation: Christopher Habel, Wolfgang Menzel, and Dietrich Dörner.

Heike Tappe and Ladina Tschander worked with me in the project ConcEv and forced me to explain and defend my ideas. Thanks also to all the students that worked in the project ConcEv over the years and contributed their time and ideas: Andy Adiwidjaja, Christian Gerhard, Steffen Huber, Nadine Jochims, Bärbel Rieckmann, Jorgen Schaefer, Heidi Schmolck, Anni-Yasmin Turhan, and Jette Viethen.

Thomas Barkowsky, Carola Eschenbach, Ralf Klabunde, and Frank Schilder read parts of this text without being obliged to do so. I am obliged to them.

Lars Kulik unfortunately left Hamburg before I had a serious version of my thesis I could bother him with, but we discussed some tricky problems, which I understand much better now.

Dick Hudson and Gerard Kempen commented on and clarified issues regarding their own work.

I also thank Cordula Klein. Mainly for standing me and leaving me alone when I brooded over something intricate while always confirming me I am not (yet) mad. She also helped me a lot with English prepositions and other language-related problems.

Last but not least I do thank my parents, who made me curious about the world and supported and encouraged me not only during the last years while I conducted this investigation. To them I dedicate this work.
THINKING WHILE SPEAKING is performed by humans almost effortlessly. Although this observation is not new, no attempt has been made to capture it in a computational model that operates in a cognitively adequate manner, i.e., a model that not only produces verbal output adequate in a given situation, but that does its computations in the same way humans do it. My goal, therefore, to construct a model that accomplishes this for a comparatively simple task, which, nevertheless, is already rather complex.

After giving a general introduction in section 1.1, I will present a motivating example in section 1.2 in order to demonstrate my two main tenets. Firstly, conceptualisation serves (among other things) as mediator between perception and language production, and secondly, in order for this to be done in a cognitively adequate manner conceptualisation must be performed incrementally. Furthermore, the section outlines the steps a human or a system has to perform in order to produce a verbal description of an observed scene while the scene takes place. In sections 1.3 and 1.4 the questions are posed that I want to answer in this investigation (1.3) and the questions that I regard as relevant for the issue of conceptualisation but that I cannot treat in depth, because they would require another text of the length of this one (1.4). In section 1.5 I lay out my methodology by describing the scientific disciplines I draw upon. Since the issues of conceptualisation and incrementality are usually treated in different disciplines (psycholinguistics and informatics) and in order to make it easier to see the connections between both issues in the main text I will pick out one example for which I will show how incrementality and conceptualisation are related in section 1.6. In section 1.7 finally, I give an overview of the remainder of the text.

1.1 Thinking while speaking

This investigation is about the fact that humans think while they speak. While this observation may sound incredibly trivial, we will see that in fact it is a highly complex task, almost too complex to be handled. In fact it is one of the issues where two
new problems crop up each time one believes to have found at least a partial answer to one of its aspects. The observation that thinking and speaking go hand in hand is not new (of course). In his famous essay Über die allmähliche Verfertigung der Gedanken beim Reden Heinrich von Kleist, for example, observes the following:*

Ich glaube, daß mancher große Redner, in dem Augenblick, da er den Mund aufmachte, noch nicht wußte, was er sagen würde. Aber die Überzeugung, daß er die ihm nötige Gedankenfülle schon aus den Umständen, und der daraus resultierenden Erregung seines Gemüts schöpfen würde, machte ihn dreist genug, den Anfang, auf gutes Glück hin, zu setzen. (von Kleist 1805)

Nowadays one would cast this in different, more sober terms: utterance production commences as soon as the communicative intention is conceived, although the utterance plan is not complete. Von Kleist goes on:†

Ein solches Reden ist wahrhaft lautes Denken. Die Reihen der Vorstellungen und ihrer Bezeichnungen gehen nebeneinander fort, und die Gemütsakte, für eins und das andere, kongruieren. Die Sprache ist alsdann keine Fessel, etwa wie ein Hemmschuh an dem Rade des Geistes, sondern wie ein zweites mit ihm parallel fortlaufendes, Rad an seiner Achse. (von Kleist 1805)

Thus, after a speaker has started an utterance on an incomplete utterance plan planning and speaking go on like ‘two wheels on one axis’. In other words, thinking and speaking are tightly connected, and they are temporally closely linked. Accordingly, speaking cannot be seen isolated from thinking.

There is a long tradition originating with Aristotle (384–322 BC) of viewing speaking and thinking as the capabilities that make humans human and set them aside from all other living beings. Perhaps the vehemence with which the discussion whether both are essentially the same or whether they are different is a consequence of this. Johann Gottfried Herder (1744–1803), for example, takes one of the extreme positions: speaking is thinking aloud. Noam Chomsky’s language organs are the other extreme, because their genetic predetermination and specialisation on language processing set them apart from a notion of thinking that encompasses speaking, cf Smith (1999b) for a discussion. Wilhelm von Humboldt (1767–1835) and Wilhelm Wundt (1832–1920), to mention just two influential scientist of the 19th century commenting on this question, take intermediary positions, in which

* The English translation is roughly this: ‘I believe that many a great orator in the moment that he opened his mouth did not know what he would say. But the certainty that he would scoop the necessary fullness of thought from the circumstances and from the arousal of his mind resulting from this made him bold enough to set the beginning on good luck.’
† In English: ‘Such talking truly is thinking aloud. The sequences of notions and their relations go on next to each other, and the acts of mind for the one and the other are congruent. The language is, then, no chain, like a drag at the wheel of the mind, but like a second wheel going in parallel on its axis.’
they acknowledge the strong interrelations between thinking and speaking but take them to be separate.

Today, the discussion of the relation between thinking and speaking is often lead by arguing for or against the hypothesis proposed by Benjamin Lee Whorf (1897–1941), now commonly referred to as the *Whorf hypothesis*. This hypothesis is an extension of views held by Edward Sapir (1884–1939) and says that a human can only think what he can talk about. Thus, it is a recent form of the first extreme position. The general problem of the influence of language on thinking is also referred to as *linguistic relativity*. So, exploring the relation between thinking and speaking is my first major topic.

The second major topic is the temporal interleaving of thinking and speaking put forward so eloquently by von Kleist. This temporal dimension of thinking and speaking is one of the points in which the views of Hermann Paul (1846–1921) and Wilhelm Wundt diverged. Paul understood the production of utterances – sentences in his terminology – in a holistic way: there is a complete conceptualisation which is then expressed sequentially. Wundt agreed only partly; he argued that a sentence has a twofold nature, a simultaneous one and a sequential one:

> From a psychological point of view, the sentence is both a simultaneous and a sequential structure. It is simultaneous because at each moment it is present in consciousness as a totality even though individual subordinate elements may occasionally disappear from it. It is sequential because the configuration changes from moment to moment in its cognitive condition as individual constituents move into the focus of attention and out again one after the other. (cited after Griffin & Bock 2006)

These issue have not been finally resolved, and perhaps they never will be. I will argue in chapter 2 that the truth is somewhere in between the extremes, i.e. I assume a *thinking for speaking*, a phrase coined by Slobin (1996), which means that when humans speak they think in a particular way. But let us start with an example.

### 1.2 Incremental conceptualisation: an example

Imagine you are sitting at an airport, look out of the window while you are waiting to board, and watch what is happening on the manoeuvring area. Imagine further that you describe what you see to another person, who cannot see what you are seeing, say, a person you are talking to on the telephone. How do you accomplish this? What mental capabilities do you need, and what mental operations do you have to perform? Assume you observe the scene depicted in figure 1.1†† In this scene

* Often the name *Sapir–Whorf hypothesis* is used. I decided to leave out Sapir, because he did not hold as strong a view as Whorf. Whorf’s views are described in Whorf (1956).

†† When looking out of the window, you will have a different perspective, i.e. you will not view the manoeuvring area from a bird’s-eye perspective. The real task I will investigate here is that scenes to
Figure 1.1: Example scene: docking of a plane

A plane docks onto a gate. Four phases (labelled 1–4) can be identified and described verbally as:

1. *Ein Flugzeug fährt auf ein Gate zu.*
   ‘A plane is moving towards a gate.’
2. *Es stoppt beim Gate.*
   ‘It stops at the gate.’
3. *Der Laufgang bewegt sich auf das Flugzeug zu.*
   ‘The walkway is moving towards the plane.’
4. *Er erreicht das Flugzeug.*
   ‘It reaches the plane.’

The cognitive task I am concerned with in this investigation is how it is possible to get from visual input to verbal output while the described events take place (not afterwards). In the terminology that I will use in the following the task is a case of incremental conceptualisation. Conceptualisation is the task of producing pre-lin-
guistic, semantic representations out of non-linguistic input. The non-linguistic input stems from various sources, in particular from outside the system, i.e., from sensory input (visual, auditory, etc), but also from other cognitive systems, e.g., from long-term memory, and from inferences and deductions carried out over already existing conceptual representations. I will mainly consider input coming in from the visual system. In short, I investigate the stretch between perceived visual input and semantic output.

The attribute *incremental* means that conceptualisation and its sub-tasks are performed in a *piecemeal* and *parallel* fashion. *Piecemeal* characterises the fact that you need not see the whole scene before you can start describing it. Instead, you can start talking, i.e., describe a part of the scene, before it ends. *Parallel* means that you perform multiple things at the same time, e.g., processing the visual input, linking this input with your memories of similar scenes, and producing a verbal description for it.† Thus, *incremental* especially captures the fact that you are capable of describing what you see while observing.

The overwhelming complexity of conceptualisation can be reduced by focusing on the *data-driven* aspects of the task. This allows to concentrate on studying

1. how a newly perceived event is integrated into the internal conceptual representation of the scene,
2. which events are verbalised,
3. the order in which they are verbalised, and
4. how a semantic expression describing an event is generated.

Therefore, I assume four tasks to be the main tasks of conceptualisation. The first, *construction*, builds up an internal representation of the external state of affairs from the perceived input by using knowledge about how the world is structured. Secondly, *selection* decides upon the content to be communicated to an (only implicitly assumed) hearer with respect to the current verbalisation goal. I only consider a fixed verbalisation goal: *describe the scene you observe!* Thirdly, *linearisation* brings the selected content into an appropriate order. For instance, linearisation decides whether two phases of the scene are described in the temporal order in which they occurred or whether the order is inverted. Finally, the *generation of preverbal messages* (*pvmsgeneration*) generates a pre-linguistic (semantic) representation for the content to be verbalised. The term *preverbal messages* for semantic representations

* Opinions differ on what semantic representations are. Since I follow the approach by [Levelt](1989), I also adopt his notion of semantic representations, which correspond to the ones by Jackendoff, e.g., described in [Jackendoff](1999, 1997, 2002). These semantic representations are a special kind of conceptual representations, viz. linear conceptual representations that can be encoded linguistically. They are, therefore, more abstract than, for example, the semantic representations by [Bierwisch & Schreuder](1992).

† In general, *parallel processing* can mean two things: simultaneous processing of multiple instances of the same problem, e.g., when a search problem is split up recursively into multiple instances, and simultaneous processing of different problems. I always use parallelism in the second sense, except when this problem is discussed explicitly in chapter 5.
was proposed by Levelt (1989).

Apart from these four there are additional tasks required that I will describe only to the extent to which they are needed for these main tasks. Two of the additional tasks play a quite prominent role, though. The first one is the perceptual pre-processing performed in the pre-processing unit (PPU). It takes perceptual input data and forms simple concepts from it that are the input to the conceptualiser. Defining the distinction between perception and conceptualisation as clear-cut as this is a simplification, because there are mutual influences. This interface should, therefore, be seen as a useful assumption that must be scrutinised and refined as soon as work has progressed sufficiently. The other additional task is called the concept matcher. It is an auxiliary task of construction for accessing long term knowledge, ie the knowledge that is present independently of the conceptualisation of a scene that is observed. In particular, this task serves to access knowledge of how to build complex concepts from simpler ones.

Focussing on the data-drivenness emphasises the need for incremental processing: as soon as a new piece of information (increment) is available it is processed with respect to the current conceptual representation. Processing the scene in figure 1 in an incremental fashion means, therefore, to use an approach in which not the whole scene is observed before conceptual processing commences, eg before boundaries between the phases are identified, but input is processed as it becomes available. This is called an online setting: language production starts before the scene has been observed completely. In contrast, in an offline setting all information required for a verbal description of the scene is available beforehand, because the scene ended before verbalisation starts. The online setting makes it possible to correlate what is happening to its verbal description. This is a major advantage, because the investigation of conceptualisation always suffers from the problem that no direct observation is possible. That is, a surface modality must be employed, and what takes place on the conceptual level must be inferred.

Describing the example scene by giving verbal descriptions of its four phases already illustrates two important points: firstly, a human observer of this scene is capable of segmenting the input stream into sub-scenes (phases). This is a prerequisite for piecemeal (incremental) processing. Secondly, the sub-scenes are part of the overall scene, ie the representation is hierarchically structured. The latter is crucial for the human observer to be able to recognise the succession of the four phases as a scene. Besides the part-of hierarchy there is another one of equal importance, the subsumption hierarchy, cf figure 2. The former hierarchy establishes the relations between an entity and its parts, eg a cockpit is part of a plane. The latter hierarchy relates kinds of entities, for which reason it is also called is-a hierarchy, because, for example, a helicopter is a kind of aircraft. The differences

---

*I will indeed make a stronger claim by extending this to the production of output as well, ie output is produced as soon as possible. I call this [Extended Wundt’s Principle](#).*

† *When I refer to concepts in an informal sense I will use intuitively plausible names for concepts, like scene. However, these labels are not used in the formal conceptual representations. The actual notation will be described shortly in section 4.4 and introduced completely in section 4.3.*
1.2 INCREMENTAL CONCEPTUALISATION: AN EXAMPLE

Figure 1.2: A part-of hierarchy (1) and subsumption hierarchy (2)

Figure 1.3: Example scene: representation of the situation structure

of these hierarchies and their importance for conceptualisation will be explored in more depth in chapter 4.

These two hierarchies are used to establish the relations between concepts, the building blocks of conceptual representations. A concept is a system-internal representation of a concrete or abstract entity or a set of entities.* What concepts are involved in our example scene? To answer this question, I first need to make a refinement. Up to now, I spoke as if scenes and phases were unanalysed wholes. However, it is advantageous to represent them in different, interrelated sub-structures. First of all, a distinction of object and situation structure must be made. This is a subdivision within the subsumption hierarchy. A part of the subsumption hierarchy representing the knowledge about how objects are hierarchically organised is given in the right half of figure 1.2. A part-of hierarchy for the situations of the example scene is shown in figure 1.3. The left-to-right ordering represents temporal precedence. Here, two types of situations are used, one for a change of position (move) and one for the transition of motion to standstill (stop). (The transition from standstill to motion is captured by a start concept, which we can leave out for the moment; but see section 4.5.) It is important to keep in mind that concepts are entities internal to the cogniser. This is of interest for situations, because there

* This use of concept differs slightly from the standard use in psychology, cf also the remark on pages 59–60.
is no consensus on whether situations are internal or external entities. Yet, since it very much depends on an individual's experience on what constitutes a situation, cf [Avrahami & Kareev 1994], I take them to be internal. Apart from objects and situations a third kind of concepts is needed to represent the scene: spatial entities.

Let us look at each of the three kinds. Concrete objects in the world (vehicles, buildings, etc) are the most obvious kinds of concepts required. In our scene we have three objects: plane, gate, and walkway. Concrete objects are related with the help of spatial entities, which provide information about where an object is located or about the path of motion it is following. Examples in figure 1.3 are towards and at. In what follows I will mostly use paths (the trajectory an object is following) and locations (the position of an object).

Each phase of the scene is represented by a situation. Situation is a term comprising notions like event, process, and state [Davidson 1967, Bach 1986]. However, distinguishing these and similar situation types is based on linguistic analysis, and making the distinctions already on the conceptual level proves difficult. Consider the first phase of the scene. As the plane is moving steadily, the situation is of the type process. Yet, firstly, the observer is capable of inferring that the movement has a starting point and an endpoint, which he did not (yet) observe but which would classify the situation as event. Secondly, after phase 2 the movement is an event, because it has an endpoint. Nevertheless, if the movement itself is described after phase 2 (The plane was moving towards the gate.) it is a process again. Thus, when describing the movement different aspects of the situation can be highlighted, depending on whether the starting and/or endpoint are taken into account. Consequently, the movement is a process as well as an event, depending on the knowledge that is highlighted. It would not be a satisfactory solution to represent both. This is especially true, because computing all possibilities of all aspects of the observed situations would significantly complicate conceptualisation. Hence, it is desirable to have an integrated representation on the conceptual level that captures what is common to all views on the situation and only determine the situation type when a situation is actually verbalised. For this reason I will simply speak of situations that are characterised by further attributes, most notably a distinction of instantaneous and extended situations and the fact whether the situation is already completed. Consequently, I will use the term process as is common in informatics, ie as a means to structure systems, not as situation type. Furthermore, I will use event and situation synonymously most of the time, because most of the situations I investigate would indeed be classified as events.

Apart from encoding temporal information the main function of situations is to link objects and spatial entities. For instance, the situation in phase 1 can be represented as move(plane, towards(gate)), where the move concept relates the plane and the towards path. If we look at the situation structure of the docking scene depicted in figure 1.3 in more detail now, we see that in phases 1 and 3 an object changes its position in the world, in the other two a moving object ceases to move, or, put differently, the movement reaches its endpoint. Note that I leave the starting point of the second movement is left implicit for the time being. Using
the concepts introduced above, the underlying conceptual representation of the verbalisations (1a–d) can be expressed in propositions like:

(2) $s_1 : \text{move} (\text{plane}, \text{towards}(\text{gate}))$
(3) $s_2 : \text{stop} (\text{plane}, \text{at}(\text{gate}))$
(4) $s_3 : \text{move} (\text{walkway}, \text{towards}(\text{plane}))$
(5) $s_4 : \text{stop} (\text{walkway}, \text{at}(\text{plane}))$

Two kinds of predicates representing different kinds of events are used: move and stop. The former describes a movement along a path, the latter the cessation of such a movement. The terms $s_1$–$s_4$ are assigned to these predicates that stand for the situations representing the four phases, which makes it possible, for example, to refer back to the situation later on. The first argument of the predicates is an object, which is the bearer of motion, ie the object that is carrying out the movement; the second argument is a proposition that describes the corresponding path in case of a move and the location in case of a stop predicate.

A disadvantage of representations given this way is that they do not explicitly state that plane, gate, and walkway are always referring to the same entity. One way to mark this is to give them subscripts, eg plane$_1$. The representational formalism should, therefore, provide means to capture the identity of entities without additional inferencing mechanisms. This means that not for each phase of a scene a new representation is created for which the relations to the previous one are then explicitly computed. Furthermore, the formalism must provide means to represent changes, eg the fact that the plane is at a different locations. So, for incremental processing it is a vital requirement that a representation can be modified and extended. Representations allowing this are called dynamic representations.

On such a dynamic, conceptual representation of the scene the four tasks introduced above execute the following operations. Construction builds up the hierarchical structure and determines that the sequence of the four phases constitute a docking situation. Selection chooses which of the situations are described verbally, eg it may decide to verbalise phases 1 and 2 and the overall docking situation. Linearisation can, for example, invert the verbalisation of phases 1 and 2, which can result in an utterance like The plane stopped after it moved towards the gate. PVM-generation, finally, generates semantic representations that describe a situation. These representations look similar to the ones given in (2)–(5).

Before I begin discussing my research questions let me make one final observation regarding the example scene. Considering figure 1.3 again, we can see that the first two motion events are performed by the plane, the last two by the walkway. This allows to group the two movements of the plane, and analogously, the two movements of the walkway. In this way we obtain a more complex and more
abstract motion concept, called simply move here, cf figure 1.4. Regarding the flexibility of cognition one can assume that humans are capable of using both kinds of representations. However, I will only use representations of the kind shown in figure 1.4 because the intermediary level will prove useful, as we will see later in this investigation.

1.3 Research questions

In this investigation I want to shed light on the issue of incremental conceptualisation for language production by pursuing the research questions that I will address in this section. Part A illuminates the task of conceptualisation. Since conceptualisation in general is too wide an issue I will restrict it by asking:

**What is event conceptualisation for language production?**

This is a threefold restriction of the task of conceptualisation. Firstly, no other modalities apart from language are considered. Secondly, I only take the production of language into account, ie the direction of conceptualisation is fixed: from perceptual input to semantic structures. Thirdly, only the conceptualisation of events (situations) is explored; objects and other entities are used only to complement event structures. Apart from these major restrictions there are some minor ones. Firstly, no reasoning takes place with elements retrieved from long-term memory. The items stored in a reduced model of long-term memory are only used to construct hierarchical structures in the conceptual representation. Secondly, no learning issues are considered; all knowledge apart from the perceptual information about the scene to be conceptualised is present from the start.

Conceptualisation is performed incrementally. Since the influences of incre-

* This is the basic idea for using event threads, cf chapter 12 which are particularly useful when dealing with concurrent events.
mentality on conceptualisation are substantial, incrementality is my second major topic, discussed in part B. Hence, the second research question is:

**What is incrementality?**

Since incrementality is just a mode of processing and, therefore, a rather general topic, my goal is to give a correspondingly broad account, while concentrating on two more special issues:

**Which dynamic representations can be used for incremental processing?**

In order for representations to be usable in incremental processing they must fulfil certain requirements. Most importantly, they have to be dynamic representations, which has effects on conceptual representations. For example, perspectivisation is no separate, transformational process, but the perspective taken is an effect of conceptualisation; more precisely: an effect of the concept utterance production starts with, cf section C. The second issue is:

**How can incrementality be used for saving resources?**

The main motivation for developing incremental methods of processing was to save resources, in particular methods that are capable of computing new information in relation to already existing information. A setting where only part of the available information changes is advantageous – in terms of resources – not to repeat all of the computations that were already carried out, but to update only those information that is affected by a change.

So, the two central notions are conceptualisation and incrementality. Until they are described in more detail, it will be helpful to keep the following formulations in mind: **incrementality is the piecemeal and parallel processing of information**, and **conceptualisation are the steps performed by a speaker to generate preverbal messages**.

After exploring both issues in their own right in the first two parts, they are brought together in the model INC (incremental conceptualiser) in part C. The discussion in this part focusses on the question

**How does the model of an incremental conceptualiser look?**

This question should be understood mainly as the question of how a model of a conceptualiser (a computational device carrying out the conceptualisation task) is constrained by the incremental processing method. Note that I do not consider this a limitation. It is rather a way to learn something new about human language production, because humans produce language incrementally and non-incremental accounts are limited in their cognitive adequacy. So, this is the point where we must ask whether empirical data of observed human behaviour actually supports INC. Thus, the last of my research questions is:

**Can INC account for the empirical data?**
The comparison of model and empirical data can be carried out with the help of the implementation of iNC. The goal is to compare transcriptions of human verbalisations and the output of the implementation. iNC must be able to account for a range of data, in particular for a default case in which the majority of cases is captured but also for unusual verbalisations, eg very detailed or superficial descriptions of a scene. Yet, since I am mainly concerned with modelling issues, the amount of empirical research is rather small, especially compared to fully-fledged psychological experiments. iNC will, therefore, have to be tested against more data. Nevertheless, the method of building the model first has a major advantage, namely that it is not in danger of giving explanations only post-hoc.

1.4 Questions that are only treated in passing

Since conceptualisation is a very wide issue, I cannot treat it in its full extent. Therefore, I will shortly sketch some issues that are relevant for investigating it that I cannot elaborate on. My assumption is that leaving out these issues does not substantially change the way conceptualisation is to be modelled.

**Intentionality / Consciousness.** Intentionality and consciousness are very tricky issues. Take the example of the selection of the content to be verbalised. It is quite natural to express the process of content selection in phrases like *The speaker decides to verbalise this concept* or *I decide to verbalise this concept.* Formulations like these resort to conscious or volitional decisions. Explaining this is, of course, far beyond what I can do in this text. Instead, my goal is to investigate the data-drivenness and incrementality of conceptualisation and to show that a lot insights can be gained by making this reduction. In other words, I assume that the volitional decision to describe a scene has been made and focus on the ensuing problems. In other words, I exclude ‘the big question’ and concentrate on how communicative intentions can be determined that serve to reach the overall goal, viz to generate a verbalisation of an observed scene in an online fashion.

**Dialogue.** Dialogue is an important aspect of conceptualisation, because it means that a speaker is really an interlocutor, ie interacts with a hearer (so, the speaker sometimes is a hearer). Therefore, the speaker refers to information he has obtained from interlocutors that now are hearers. Focussing on the data-driven aspects of language production is, again, a reduction of complexity, because the dialogue is reduced to a monologue: no information must be taken into account that stems from other interlocutors. Additionally, other interlocutors are only implicitly assumed but never modelled. This is tantamount to assuming a default hearer. The reduction to a monologue setting also means that only some aspects of *discourse* have to be considered, above all that a generated utterance is consistent with previous and future utterances. In other words, the fact that the speaker produces not only one isolated utterance but sequences of utterances is paramount.
Multimodality. Conceptual representations should be general enough so that they cannot only be used for speaking but also for other tasks in other modalities. For example, a conceptual representation generated in the airport scenario described in section 1.2 cannot only be used for speaking but also for navigating through space. Navigating through space would require either another conceptualiser – operating on the same conceptual representation – or a conceptualiser that is also capable of thinking for navigating in addition to Sobin's (1996) thinking for speaking.

However, I do indeed consider multimodality in another respect. In my setting the input of the conceptualiser is non-linguistic; it is input from the visual system, and the overall task of conceptualisation is to translate it into propositional semantic representations. In this view conceptualisation is comparatively independent from linguistic considerations; rather it is a mediator between language and other modalities, cf also section 2.2.

Gestures. One issue of multimodality is especially relevant for conceptualisation, the generation of gestures. There exist proposals of how the language production model by Levelt (1989), on which this investigation is based as well, can be extended in order not only to generate language but also speech accompanying gestures. Two of these models are described in Krauss, Chen, & Gottesman (2000) and de Ruiter (2000). Both models are not implemented and let it be enough to say that both do not fit to C (although the latter would be suited better). Nevertheless, for the production of an utterance the verbal and the gestural output originate from a common source, termed growth point by McNeill (1992). And although McNeill himself totally rejects the idea of an information processing approach (McNeill 2000) something similar to growth points must be located within the conceptualiser, because otherwise gesture and utterance could not be about the same content, and they could not be generated in a coordinated fashion, eg with respect to time.

The fact that gestures are additional means of expressing content suggests that they are part of conceptualisation. They can, however, be regarded as additional features of a conceptualiser model. The main reason for this assumption is that Levelt’s as well as Krauss et al’s approach both do not take into account that only after content to be verbalised has been selected and linearised the planning of accompanying gestures can commence. That is, only during the generation of the semantic representation the corresponding gestures can be planned. Thus, the conceptualisation of utterance and accompanying gesture proceeds undivided until rather late in conceptualisation, namely just before the generation of the preverbal message begins. Part of the problem seems to be that notions of communicative intentions of de Ruiter as well as Krauss et al differ from Levelt’s.

† According to McNeill information processing approaches suffer from the neglect of context. With this he means that due to the modularity used in information processing approaches the generation of linguistic and gestural output cannot proceed in a synchronised fashion. In his view the synchronised movements of body parts performing gestures and the articulation of speech is only possible if this is done within the same context, where both modalities can influence each other. Since such arguments are not new (and it is never made clear what context should be), a lot could be said about this, which I will not do here. The issue of modularity is extensively dealt with in chapter 2 and the notion of quasi-modules discussed there are one solution for McNeill’s problems.
claims that his language production model is not particular to speaking, for which it was developed, but could also be used for writing. Similarly, an extension by gestures seems possible. Therefore, I will concentrate on the core functions. Observe that for successful communication via writing no gestures are required either.

1.5 Disciplines and methods

I draw on methods and results from different disciplines: cognitive science, artificial intelligence (AI), linguistics, natural language generation (NLG), and natural language production. Let us look shortly at what each of these contributes to the overall undertaking.

The aim of cognitive science is to build models and systems that mimic (simulate) human behaviour as closely as possible. In doing so the main aim of cognitive science is to explain the way that processing is done, not just to build models that match input to output in order to reproduce empirical data. I will use the terms model and system as follows.

Term 1.1 Model. A model is a textual and/or formal description of tasks and representations.

Term 1.2 System. A system is an implemented model. In order to build an implementation (an executable program) additional assumptions have to be made, while other issues can only be realised in a very reduced version due to complexity.

According to this distinction, cognitive science is more concerned with the creation of models than of systems, because the main aim is to understand how the mind works – to pick up the title of Pinker's (1999) book – and not simply to construct a running program. INC is designed with the aim to account for the processing mechanisms that are employed by humans in order to accomplish the task described in section 1.2. Hence, it has mainly the status of a model. There are two reasons for also implementing INC, ie for building a system for the model. Firstly, with the implementation it is possible to compare the behaviour of the model with verbalisations produced by humans. Secondly, since implementing always means to make further decisions in order to make the model detailed enough to be executable, modifications of the model are required, which improves its quality and accuracy. However, building a system also means that not all issues can be realised as detailed as would be desirable because of the resulting complexity of the system. For example, although there are sophisticated models of human memory, INC only contains comparatively simple kinds of memories. Making them more detailed would increase the complexity of INC, while not contributing to the core ideas for whose exploration INC is built; there would be too much effort for only little gain.

Throughout this investigation I will provide short characterisations of central terms. One might also call them definitions, but I want to avoid the mathematical connotation. Nevertheless, in the running text I will say define and definition as well. This is also not meant in the mathematical sense.
INC realises a part of the model of language production by Levelt [1989, 1999]. This model comes from the field of natural language production, which is a sub-field of psycholinguistics. Psycholinguistics is a discipline contributing to cognitive science, and the overall goals of both are the same.

Since the output of conceptualisation consists of semantic representations of utterances, results from linguistics are used to check whether the representations are adequate not only under a processing perspective but also under a descriptive-structural one. This is necessary for two reasons. Firstly, the body of linguistic data investigated in linguistics is considerably larger than the one in psycholinguistics, and psycholinguistic data about semantic or even conceptual phenomena are even more scarce. Secondly, the data collected in psycholinguistics and linguistics are rather different. Psycholinguistics uses mainly reaction time experiments or other psychological methods while linguistics is concerned with explaining differences in the structure of (a) language. A model, however, should be capable of explaining both kinds of data.

From artificial intelligence (AI) in general and natural language generation (NLG) in particular – seen as a sub-field of AI – come techniques for building a model that conforms to cognitive constraints and for refining the cognitive model in order to produce a executable system. In the cases where there are gaps in the model one can use technical solutions, because natural and technical solutions are often similar. The underlying assumption is that the task has to be computed somehow – by humans as well as by a system – and that the computations must be similar.

To sum up, even though there are differences in the goals of cognitive science and NLG/AI, they are far from being incompatible. On the contrary, there is a considerable overlap, which can be seen by the fact that AI is usually regarded as one of the mainstays of cognitive science. Systems for the generation of natural language built with and without cognitive considerations have a lot more commonalities than differences as a comparison by Reiter [1994] shows, cf also De Smedt, Horacek, & Zock [1996] and Reiter & Dale [2000]. Nevertheless, transfers from NLG/AI to cognitive science can only be done with great caution, because doing this must not limit the cognitive adequacy of the resulting model and/or system.

The remainder of this section is structured as follows. I will first take a cognitive science, then an NLG/AI perspective on the task described in section 1.2. After that I will outline what can be learnt from bringing the two perspectives together.

The cognitive science perspective. My major aim is to account for the way humans accomplish the verbalisation task described in section 1.2, which I will, henceforth, refer to as online description of events. The means to provide explanations is building the model INC, which should be as cognitively adequate as possible. Yet, cognitive adequacy is not only to be achieved for the overall behaviour of the model but also for the way in which the computations are carried out, i.e. the way in which the sub-tasks of conceptualisation are performed.

As already pointed out in section 1.2 the overall task can be subdivided into two major parts: a perceptual pre-processing and the conceptualisation proper. I re-
gard the perceptual pre-processing only as subsidiary task in order to get from the
input of the overall system to simple concepts (perceived entities). The four main
tasks of conceptualisation were already mentioned as well: construction of a concep-
tual representation, selection of events to be verbalised, linearisation of the selected
events, and the generation of preverbal messages (pvm-generation) describing the
events. Let us look shortly at each of these tasks. In the first task, construction, the
perceived entities are taken as input, and more complex (simpler) concepts are con-
structed by a grouping (segmentation) operation. In the example from section 1.2
the simpler concepts are taken together to form the complex docking and move
concepts. The second task, selection, decides which event(s) of the conceptual re-
presentation are verbalised. Since I am dealing with the conceptualisation of events,
the selection task only examines the event (situation) structure. Thus, its decisions
are about whether the complex docking event is verbalised and the simpler events
are not, but other concepts, for example objects, are not even considered for verbal-
isation. The third task, linearisation, brings the selected events into an appropriate
order. This task is important in contexts in which it is advantageous to generate
utterances that do not reflect the chronological order, for example

(6)  *Das Flugzeug hat angedockt, nachdem es gelandet ist.*
‘The plane docked after it landed.’

The last task, pvm-generation, generates a suitable preverbal message (a semantic
representation) for the selected and linearised content. This is necessary, because
an event concept like docking is no semantic representation. Therefore, it must be
decided how it is verbalised. For example, describing this event involves generating
a referring expression for the plane that is the bearer of motion, eg *das Flugzeug* ‘the
plane’, *das große Flugzeug* ‘the red plane’, or *es* ‘it’. The decision of what constitutes
an adequate description of the plane has not been made by selecting the event.

The execution of these tasks as well as the performance of humans and systems
in general depends very much on the available resources. As most resources, cogni-
tive resources are limited. Therefore, a cognitive model must account for these limit-
ations. This is particularly relevant when building a system for a model, because the
limitations of resources are not (should not) be subject to technical considerations
(does the computer have enough memory?) but subject to cognitive considerations
(what happens if human memory is exceeded?). InC allows to vary the available
resources by assigning values to parameters, eg for the number of events that can
simultaneously be selected for verbalisation. Although conceptualisation is mainly
about how an individual human conceptualises the world, there are also general
(intersubjective) mechanisms that enable humans to perform the task at hand. The
setting of resource parameters proves to be a useful mechanism to achieve this: the
majority of human verbalisations of a scene can be captured by setting the para-
eters to default values, while more unusual verbalisations need special settings.
The NLG/AI perspective. While my overall goal is to build a cognitively adequate model of an incremental conceptualiser, many a technique employed in this undertaking is taken from AI and NLG. The aim of NLG/AI is to build systems that behave in an intelligent way (AI) or that generate natural language (NLG). Although the issue of cognitive adequacy pops up now and then it is not the main concern.

Since the study of conceptualisation suffers from the problem that it is only indirectly observable, the empirical data inevitably have big gaps. This is even more so, because only very few empirical studies on conceptualisation exist. Thus, although iNC should integrate as many results from empirical studies as possible, 'white spaces' remain. When filling the gaps in the model, one has to estimate how humans solve the problem. These estimates can and should be used as hypotheses for further empirical investigations, but for the model it unavoidably means that these parts may be empirically wrong. This problem strikes even more forcefully when the model is refined into an executable system. Thus, due to the proposed similarity of computations by humans and artificial systems techniques that proved their usefulness in NLG/AI can be employed for filling the gaps.

I draw on results from different areas of AI: firstly, the processing of sensory input, secondly, representing knowledge, which are both well-studied in AI; thirdly, lying at the core of this investigation, the issue of dynamic representations, ie questions of how knowledge can change over time. Finally, as a sub-discipline of informatics AI provides means for dealing with everything that has to do with modelling, system-building, and embedding a model in an environment.

There are, however, also significant differences to the NLG/AI approach. The four tasks described above have no real counterparts in typical NLG systems. The most important reason for this is that such systems are mostly built for text generation, not speaking. Consequently, incremental systems – where subsequent planning stages are carried out in parallel, output is generated while input is read in, and which usually operate without feedback – are very rare. Typical NLG systems perform the following tasks for the generation of a semantic representation: content determination, document structuring, aggregation*, and generation of referring expressions, cf [Reiter & Dale (2000)]. (Lexicalisation is sometimes included in this list.) As can be seen, the division of labour is quite different compared to the one proposed above, where, for example, the generation of referring expressions is part of PVM-generation, not a separate task.

iNC also differs from typical NLG approaches (and some strands of AI as well) in that it is embedded in an environment to which it reacts. The ensuing data-drivenness is particularly important for the time it has available for its computations. Since iNC shall be capable of generating output in (almost) any amount of time, it has some similarities to anytime approaches, cf also section 7.3.

---

* Aggregation means information is expressed in one swoop and not in multiple utterances. An example is that instead of *A man crosses the street* and *The man has an umbrella* only one utterance is generated: *A man with an umbrella crosses the street.*
Bringing the views together. To sum up, my aim is to employ AI techniques in order to achieve the overall goal of cognitive science, viz explaining (human) cognition. The means to do this is to build a cognitively adequate model, an approach best described as cognitive modelling. It goes back to the works by Allen Newell and Herbert Simon, described, for example, in Newell & Simon (1963, 1972). Why is this worthwhile pursuing? First of all, cognitive science profits from models of cognition, because such models propose hypotheses of how the mind works. Corroboration as well as falsifications of such hypotheses serve to gain new insights.

One could also say that cognitive modelling is for AI what bionics is for engineering. Opposing such a view, Russell & Norvig (2003) bring forward the example that aircraft could be constructed only after their builders abandoned the idea of building artificial birds. Analogously, AI should not bother with trying to build artificial humans. Yet, the air streaming along the surface of a wing works according to the same physical principle in birds and aircraft. So, although in those days nobody spoke of bionics, the transfer of ideas is similar. Furthermore, as Russel and Norvig say themselves, in particular in the fields concerned with the processing of natural language a collaboration between cognitive and non-cognitive approaches proved fruitful, eg, the generation of referring expressions from the field of NLG (sections 3.4 and 13.2). There is good reason for this: for language (in contrast to flying) humans set the standard, ie humans define, for example, what counts as an adequate description of a scene. Thus, for the online description of events one can stipulate that there is no difference between an intelligent system and a human in performance as well as in the way they do the processing.

Put concisely, from cognitive science I take the goal to produce a model /system that is as cognitively adequate as possible; from NLG/AI I take methods to refine the model and methods to build a system, which can then be tested against empirical data. The results can be used to refine the model, and the circle starts anew.

1.6 Effects of incrementality on representations

Representations are an important issue when building a computational model. To facilitate argumentation in the following chapters, I provide the representational groundwork in this section. The main point of this section is that in the context of conceptualisation it is a stark abstraction to stipulate representations without taking the time factor into account – an abstraction that leads to artefacts. I will demonstrate this for the case of the perspectivisation of utterances. Accounts that do not consider the dynamics of cognitive processing in their representations, ie the changes of the representation over time, must assume a separate perspectivisation process. In an incremental model this is not necessary, but perspectivisation is just an effect of the time course of utterance production. However, before that I will give a short account of the representational formalism I use, referential nets. A more detailed description will be given in section 4.3.
Short overview of referential nets. Referential nets (refNets) consist of interrelated referential objects (refOs, \[Habel\, 1982,\, 1986;\, \[Eschenbach\, 1988\]. A refO represents an entity. Formally, a refO is a term \((r_1, r_2, r_3, \ldots)\). A refO can have two kinds of information associated with it: attributes and designations. An example of a refO together with its attributes and designations in the standard notation is:

\[
\begin{array}{c}
\text{human} \\
\text{male} \\
\end{array}
\xrightarrow{r_1}
\begin{array}{c}
\text{DAVID} \\
\text{father_of('RUTH')} \\
\text{ix wife('SARAH', x)} \\
\end{array}
\]

Attributes are written to the left, designations to the right. Attributes represent conceptual knowledge like the sort of the refO, which is mandatory and always stands in first position, here \(\text{human}\). Other attributes contain essential, defining properties of the entity that can be used for conceptual inferencing. Designations are meaning-related expressions of one of three kinds: \textit{names} ('DAVID'), \textit{functional expressions} (father_of('RUTH')), or \textit{descriptions}. Descriptions are of the form: \(\text{op var pred}\), where \(\text{op} \in \{\wp, \eta, \text{all}_t, \text{some}_t\}\) is the operator, \(\text{var} \in \{x, y, z, \ldots\}\) a variable, and \(\text{pred}\) a predicate–argument structure. The operators reflect the cardinality of the refO and the definiteness of the predicate (\[Habel\, 1986\, 137\], cf table 1.1). Thus, \(\text{ix wife('SARAH', x)}\) can be read as the entity whose wife is Sarah. Correspondingly, \(\eta \text{ x wife('SARAH', x)}\) stands for Sarah is a wife of this entity – in which case the entity represented by \(r_1\) (David) could have more than one wife.

<table>
<thead>
<tr>
<th></th>
<th>DEFINITE</th>
<th>INDEFINITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cardinality = 1</td>
<td>(\wp)</td>
<td>(\eta)</td>
</tr>
<tr>
<td>cardinality &gt; 1</td>
<td>(\text{some}_t)</td>
<td>(\text{all}_t)</td>
</tr>
</tbody>
</table>

Table 1.1: Operators in referential nets

RefOs, attributes, and designations can be added, deleted, or modified. Therefore, referential nets are dynamic representations. This property and the fact that they are well-suited for representing conceptual and meaning-related knowledge are the main reasons why I use them. Furthermore, referential nets can be extended in order to represent multimodal knowledge. \[Habel\, 1987\] extends them by introducing depictions for representing pictorial knowledge about the refO. Similarly, it would be possible to extend the formalism by knowledge in other modalities.

Perspectivisation as an effect of incrementality. Traditional (non-incremental) accounts of language production usually assume a perspectivisation task in its own right (\[Ziesche\, 1997\]). It arises out of the necessity that an utterance must have a perspective. A typical example is:

\[
\begin{align*}
\text{(8) a. } & \textbf{Mary hits John.} \\
\text{b. } & \textbf{John is hit by Mary.}
\end{align*}
\]
Both utterances, it is assumed, describe the same state of affairs but have a different perspective. In other words, both are statements about the world that describe the same proposition: hit(mary, john). Yet, (8a) is a statement about Mary while (8b) is a statement about John. The standard view on this issue is that it is one decision to verbalise the state of affairs, i.e., decide upon the proposition, and only then to decide upon the perspective.

There are, however, two different explanations. The first is that (8a) and (8b) are generated from different conceptualisations, i.e., there are different propositions on the conceptual level: hit(mary, john) and hit_by(john, mary). In this case the perspective taken depends strongly on perceptual factors, e.g., whether the observer focusses on John or Mary when the scene occurs. It is quite reasonable to assume that the situations conceptualised differently, because hitting and being hit are quite different things – as you will know of your own experience.

The second explanation is based on processing considerations, and is therefore the interesting possibility with respect to incremental processing. The above proposition hit(mary, john) can be represented in a referential net as follows:

\[
\begin{align*}
\text{human} & \quad r_1 & \quad 'MARY' \\
\text{female} & \quad \eta_x \text{hit}(r_2, x, r_3) \\
\text{situation} & \quad r_2 & \quad \eta_x \text{hit}(x, r_1, r_3) \\
\text{human} & \quad r_3 & \quad 'JOHN' \\
\text{male} & \quad \eta_x \text{hit}(r_2, r_1, x)
\end{align*}
\]

This representation contains three refO's, one for each person and one for the situation. They are related via a description consisting of hit as predicate and three arguments: the situation in first, the 'hitter' in second, and the 'hittee' in third position. The latter two correspond to the thematic roles agent and patient. Thus, these expressions are already close to a semantic representation. On the basis of this representation the perspective taken hinges on the sequence in which the refO's are accessed. First of all, the speaker may start with one of the refO's representing Mary (r_1) or John (r_3). In this case the perspective is already taken, because the speaker makes a statement not about a state of affairs in the world (at least not primarily) but about a person, Mary or John. Alternatively, the situation refO (r_2) can be chosen first. Then, the perspective is a matter of which of the other refO's is accessed first, Mary or John. Note that referential nets also allow to use different predicates for hit and hit_by, which can, for example, be represented by two descriptions on the situation refO r_2.
1.7 Outline

The text consists of three major parts. Part A addresses the issue of conceptualisation. Chapter 2 will draw the boundary between language production and other cognitive tasks and describe the language production model by Levelt (1989) of which inC models the first component. Chapter 3 is devoted to the discussion of conceptualisation and the sub-tasks that must be performed by a conceptualiser in order to generate online descriptions of events. These sub-tasks constitute the major division of labour that is used by inC. Chapter 4 describes the representations that are used for accomplishing the conceptualisation task.

In part B I will give a general account of incrementality. In the first chapter of this part (chapter 5) I will present a general account of incremental processing by discussing relevant literature, providing a blueprint of incremental models, and pointing out the dimensions along which incremental models can vary. Chapter 6 contains a formalisation of the notions introduced in chapter 5 as next step towards creating the model of an incremental conceptualiser. Chapter 7 provides a first integration of the first two parts by discussing the role of resources for incremental conceptualisation and the reasons why I did not use an existing unified cognitive architecture, like Soar or ACT-R, for modelling inC.

Part C presents the inC (incremental conceptualiser) model that brings together the issues of the first two parts. After describing the overall architecture in chapter 8 inC’s representations and processes are the topic of the ensuing chapters. Examples of simulations are reported in chapter 14 and in chapter 15 I outline an extension of inC by a rudimentary monitoring component.

The last two chapters are devoted to suggestions on how inC can be extended to model other factors influencing human conceptualisation (chapter 16) and to an overall conclusion (chapter 17).

There is one important point about the structure of the text. Part A abstracts away from incrementality, while part B abstracts away from conceptualisation. (For this reason the transition from chapter 4 to chapter 5 is a bit rough.) Nevertheless, I will constantly comment on the connections between the two issues.
CONCEPTUALISATION
LANGUAGE PRODUCTION

Language production is only one of a plethora of cognitive tasks humans perform. This chapter has, therefore, two main topics: firstly, to give an outline of language production, secondly, to describe how it is related to other cognitive tasks and how it can be set apart from them. Put differently, I want to draw a boundary between language production and other cognitive faculties. Drawing such boundaries is a common problem in cognitive science and is closely related to the problem of modularity. Therefore, I will discuss the degree to which the human language production faculty is modular.

On the one hand language production is a task in its own right, ie separate from other cognitive tasks, while on the other hand it intermingles with other tasks to a degree that no clear boundary can be drawn. This intermingling is particularly true of the conceptualisation part of language production. Nevertheless, language production in general as well as conceptualisation in particular possess important traits of a module. Since I am concerned with conceptualisation, I will focus on issues relevant to conceptualisation.

After motivating that it is worthwhile to investigate language production in section 2.1, I will elaborate on the mediator function of conceptualisation in section 2.2 and present Levelt’s (1989) language production model in section 2.3. In the remaining three sections I will then illuminate to which degree the human language production system is modular. I will first give criteria in section 2.4 that allow to judge whether a component is a module and explain the idea of quasi-modules. In section 2.5 I will argue that some problems in investigating conceptualisation that are due to non-modular properties can be overcome by concentrating on the data-driven aspects. Finally, I point out in section 2.6 that conceptualisation – being the least modular component of language production – is somewhere between being modular and being non-modular by discussing the degree of language specificity of conceptualisation. My conclusion is that conceptualisation for language production is best characterised as thinking for speaking (Slobin 1996).

Please note, again, that incrementality plays only a minor role in this part. The discussion of incremental processing will follow in part B. Conceptualisation and incremental processing are then combined in the iNC model in part C.
Investigating language production

Compared to the number of models and systems for language comprehension there exist only few for the production of language. Despite the growing interest in automated language processing in recent years this situation has not changed. Only ca 10% of pages in textbooks on computational linguistics, for example, are about language generation [Jurafsky & Martin 2000, Carstensen, Ebert, Endriss, Jekat, Klabunde, & Langer 2001]. In addition, most language production systems have only very little to say about how humans produce language, because although they are designed for generating language that sounds as naturally as possible they do not perform the computations in a cognitively adequate manner. For an outline and an overview of non-cognitive systems see Reiter & Dale (2000).

One reason for the small number of language production systems is that the task of generating language is often (albeit often implicitly) regarded as the inverse of language comprehension. The assumption is that once language comprehension is fully understood one simply inverts the model/system to get a language production model/system. However, parsing, the basic method used in language comprehension systems, plays only a minor role in language production, while very different methods are important for language production. McDonald (1987) illustrates this point by characterising the generation of natural language as being mainly a choice task: after the content to be verbalised has been decided upon, a sequence of choices must be made in order to find the linguistic means that are suitable to express the chosen content. In contrast, language comprehension mainly consists of hypothesis management: given a sequence of words one must hypothesise what meaning fits the parsed material best. The initially infinite set of hypotheses is narrowed down step by step.†

Additionally, research not motivated by cognitive but practical considerations can find surrogates for most language generation tasks, eg the use of pre-fabricated, ‘canned’ text. This is due to the fact that for most practical applications it is possible to define a set of possible expressions sufficient for the task. Corresponding methods are regarded unacceptable for language comprehension systems in most cases, because there the goal is to build systems that can cope with the full complexity of language. A method to reduce the complexity, which, however, is most often is not even considered seriously, is to use so-called controlled languages, ie languages that are restricted in their expressive power. Furthermore, the choices during language production

As I already pointed out in section 1.5 there is a distinction between the field of natural language production and natural language generation (NLG). The former is the field that aims at cognitive adequacy of the model/system while the latter does not. I do not make this distinction and use production and generation interchangeably; I always aim at cognitive adequacy. The cases in which I use results from NLG will be indicated in the text.

† Viewed from the perspective of incremental processing, the first task is a sequence of choices, each one dependent on the former choices, while the latter is a piecemeal narrowing down of the set of possible meanings until the best candidate for the meaning is reached. Note that this distinction serves only to motivate the overall point. It is not meant to define the two tasks.
production are often regarded as less critical operations than the removal of a hypothesis in language comprehension. The main argument is that the removal of a hypothesis may have the effect that the system is not able to come up with the correct meaning of an utterance. However, in language production almost every choice can result in that it is impossible to make subsequent choices, which are necessary for expressing the chosen content.

Taking this together with the arguments presented in chapter 1, there are three main reasons for investigating language production. Firstly, compared to language comprehension and to the non-cognitive approaches to language production (NLG) only very little research has been conducted. Secondly, language production viewed from a cognitive instead of a technical perspective offers the chance to learn from nature, providing additional methods, which can be used for building better NLG systems. Thirdly, the problems of language production are different from the problems of language comprehension: inverting comprehension will not yield models of language production. For building natural dialogue systems interacting with a human user this has, for example, the consequence that it does not suffice to build a sophisticated language comprehension system; one must also build an adequate language production sub-system.

In addition to this, viewing language production under a dynamic, incremental perspective, has a further advantage: it is a means to reduce the complexity of language production systems. The reason is that there always is one element in focus, i.e., one element is interpreted with regard to the current state of the system. In contrast, most NLG systems take all information that is available into account. The advantage can again be seen in the example of dialogue systems. Firstly, dialogue systems have to give quick responses; therefore, the first bit (increment) of output should be produced as soon as it is available. Secondly, as Menzel (1994) argues, in mixed initiative dialogues a system must keep track of the output of the human user in order to keep the pause as short as possible, for example, or to decide whether to interrupt the interlocutor.

2.2 Conceptualisation as mediator for language

In the production of online descriptions of events visual input and verbal output are correlated. This raises the issue whether cognition is multimodal. Obviously, the 'visual world' and the 'world of spoken language' are different, and it is very likely that they require representations in different modalities. Thus, when happenings in the one world stimulate happenings in the other, a translation must take place. This is one of the tasks conceptualisation performs.

However, the proposal that all thinking takes place in only one – this is usually tantamount to propositional – format is very prominent. Two of a number of debates about this question are the hypothesis of a language of thought and the imagery

* In part B such elements will be called focussed elements.
debate. The idea of a language of thought is proposed by Fodor (1975). According to this view humans think with an inventory of symbols that can be combined with a limited set of rules. Fodor’s argument is built on the claim that there is a strong similarity between linguistic representations and other mental representations. One point supporting this view is that propositional representations are easily capable of representing abstract entities, which is rather difficult with other representational formats. If there is a language of thought then all thinking is propositional.

However, the issues discussed in the imagery debate cast severe doubts on this account of thinking (Kosslyn 1994). Results from empirical studies indicate that imagistic representations are different in nature from propositional ones. One of the main arguments for the existence of imagistic representations comes from experiments where participants have to rotate objects mentally. The rotation takes longer the greater the angle of rotation, e.g., rotating an object by 45° takes less time than rotating it by 225°. In a purely propositional account the different angles cannot matter, because processing a symbol representing 45° takes as long as processing a symbol representing 225°. Therefore, a different kind of representation must be used for such operations, imagistic representations. Yet, while Kosslyn declares the debate resolved (Kosslyn 1994, 1995), Pylyshyn argues there is no convincing evidence against a solely propositional reasoning (Pylyshyn 2001, 2002). Pylyshyn’s main objection is that from the fact that it is possible to describe phenomena of image processing in imagistic terms it does not follow that this must be the case. He argues that if there are propositional theories offering adequate explanations, they should be preferred, because then there is no need to introduce a new theory.

Further evidence against a solely propositional account of cognition comes from Glenberg, Robertson, Jansen, & Johnson-Glenberg (1999). Their empirical results even contradict explanations that are traditionally given in propositional terms. One of these is the processing of negations. An assertion A and its negation ¬A are processed equally fast, while in the propositional account the negative version should require more time, because first A must be evaluated and only then can its negation be computed in a second step.

There are also promising attempts to integrate imagistic and propositional representations. One example is the mental models approach of Johnson-Laird (1983), which are, however, closer to imagistic representations than to propositional ones.

Debates and empirical findings like these show that it is quite likely that propositional representations alone do not suffice for a full account of thinking. Other types of knowledge that must be reckoned with are most notably spatial, imagery, and kinaesthetic knowledge (Levelt 1989: 73; figure 2.1). On the semantic level, however, all representations must be propositional, cf also the next section. Conceptualisation must therefore convert non-propositional knowledge into propositions, including perceptual information. Since the formalism referential nets, which is used for nC’s conceptual representations, is capable of representing knowledge in other modalities as well, cf section 6.6. I assume that adding modalities only enhances nC.
but does not alter it substantially.* However, the repercussions of integrating other modalities into INC may turn out to be too severe to hold up this position.

An issue equally important as modalities for an account of conceptualisation is that conceptualisation also depends on the domain. For example, Markman and Gentner state that ‘[e]ven in the seemingly abstract domain of mathematics cognitive performance is affected by domain content.’ (Markman & Gentner 2001: 224) Yet, INC was developed within two different domains, online descriptions of drawings of sketch maps and descriptions of motion events, cf also chapter 14. For this reason the claims I make possess some generality, and it can be said that they apply to conceptualisation in general, not only to a particular domain.

In the following I will focus on propositional representations and on online descriptions of motion events. For the stated reasons I regard this as sufficient.

2.3 Levelt’s tripartite architecture

There is only one prominent model of language production that is based on cognitive and psycholinguistic considerations and that furthermore goes all the way from communicative intentions to articulatory output. It is the one by Willem Levelt, described in Levelt (1989) and in modified versions in Levelt (1999) and Levelt, Roelofs, & Meyer (1999). My investigation is mainly based on the 1989 version as framework, because this is the one that makes the most elaborate claims about the conceptualiser, while the others mostly focus on lexical access. Levelt concentrates

\* INC’s perceived entities (PES) could be regarded as being already propositional instead of (still) visuo-spatial representations. This would mean that the translation takes place in the pre-processing unit (PPU). Yet, since referential nets can be used for additional modalities, this does not affect the overall argument of conceptualisation as mediator.
Figure 2.2: Levelt’s (1989: 9) blueprint for the speaker (slightly coarsened)

on speaking in his work and assumes that writing or signing’ function along the same lines, despite their different output modalities. Instead of an articulator (see below) a signing or writing component would be required, of course.

The three main components of Levelt’s architecture are the conceptualiser, the formulator, and the articulator, cf figure 2.2. The two interfaces between these components are the preverbal message between conceptualiser and formulator and the phonetic plan between formulator and articulator. The strict sequentiality of the architecture allows no feedback between components but only module-intern (Levelt 1989: 15f). Consequently, a component does not know how far its output is already processed by subsequent components. This is relevant, for example, in cases where output is erroneous or incomplete, because there is no possibility for a subsequent component to send back information about this to a preceding one. The Levelt model, therefore, contains a monitoring component that reads in the parsed speech of the speech comprehension system. Besides component-internal feedback this is

*Signing means using a sign language. This is a different communicative modality than gesturing, because gestures are no full language in their own right but usually are speech-accompanying.
the only feedback in the model. It enables the conceptualiser to keep track of what of the planned utterance(s) is already produced, and it makes possible to detect deviations. This is called self-monitoring, because it is not the monitoring of other interlocutors but of the speaker himself. Since this is only a model of a speaker, monitoring of another person speaking is considered.

Let us now look at the components in more detail. The conceptualiser consists of two sub-components: message generation, which generates preverbal messages, and monitoring. InC is a model of the message generation component. Message generation performs four main tasks (Levelt’s terms are given in parentheses):

1. building up of an internal representation (bookkeeping),
2. selection of the content to be verbalised (macroplanning 1),
3. bringing the selected content into a linear order (macroplanning 2),
4. generating a preverbal message for the content to be verbalised (microplanning).

In InC these tasks is realised as processes: construction, selection, linearisation, and preverbal message generation (pvm-generation). All processes work incrementally, ie as soon as a new input increment is available it is processed. Using processes as means of structuring allows to give another characterisation of incremental processing: incremental processing is the parallel processing of a sequential information stream.* This means three things. Firstly, there are multiple processes running in parallel. Secondly, these processes are arranged in a fixed sequence so that the output of one process is the input to its successor. Thirdly, this characterisation conforms to the no-feedback condition by Levelt.

As already pointed out in section 2.2 the conceptualiser has access to different kinds of knowledge. Besides an internal representation of the external states of affairs it has access to the discourse model, long-term memory, which contains encyclopedic knowledge, and many more; in fact, the conceptualiser has access to all information that can be expressed verbally. However, knowledge represented in non-propositional formats must be transformed into propositional knowledge in order to be verbalised, because at the end of conceptualisation such knowledge must be present as preverbal messages, the characteristic input of the formulator. For the necessity of characteristic input see section 5.4.

Preverbal messages are a special kind of conceptual representation. Mainly two properties set them apart from other conceptual representations. Firstly, they are sequential representations, while conceptual representations are hierarchical. Thus, preverbal messages contain neither part-of nor subsumption hierarchies. Secondly, preverbal messages are those conceptual representations that contain meaning to be conveyed.* Jackendoff 1987, 1990, 1997, Wiese unpublished. They hold exactly that information that is necessary for the formulator to encode the intended meaning linguistically, ie to generate a phonetic plan. This means in particular that the

* This information stream is split up into a sequence of increment streams and /or increment buffers defined on pages 91 and 96 respectively.
following components have no access to the internal representations of the conceptualiser as this would be just another form of feedback. In this sense they can also be called pre-linguistic or semantic representations. However, this view is not shared by all semantic theories, eg two-level semantics (Bierwisch & Schreuder 1992). Additionally, the traditional view on preverbal messages as static semantic representations is a simplification that is inadmissible in the context of incremental processing. Instead, one must be aware of the fact that preverbal messages are representations that are generated and processed incrementally as well, cf section 1.6. To emphasise this point I will also call them incremental preverbal messages, ie I will regard them as sequences of increments (Guhe, Habel, & Tappe 2000). Under the production perspective this has two eminent consequences. Firstly, for each increment of an incremental preverbal message it must be ascertained that it fits to the increments already generated. This is the consistency condition. Secondly, all increments of an incremental preverbal message taken together must form a complete semantic representation as it is required by results from semantic analyses, eg Jackendoff (1990). This is the completeness condition. A detailed account of incremental preverbal messages will be given in section 13.1.

Since I do not consider components of Levelt’s model other than the conceptualiser, I will only sketch out the other components. The formulator takes preverbal messages and encodes them linguistically in two stages (not indicated in figure 2.2). First the surface structure is generated by the grammatical encoding component. The phonological encoding component then produces the phonetic plan from the surface structure. The formulator has access to the lexicon. Models of the formulator include IPG (Kempen & Hoenkamp 1987), IPF (De Smedt 1990), SYMPHONICS (Günther, Schopp, & Ziesche 1995), WEBER (Roelofs 1997), and Performance Grammar (Kempen forthcoming). The phonetic plan is a program for articulation. It is called internal speech when the emphasis lies on its role as direct input to the speech comprehension system. The articulator computes motor commands from the phonetic plan and executes them. The result is the production of overt speech, which is a succession of speech sounds. These are the output of the articulator and input to the audition component.

Audition transforms overt speech (speech sounds) into a phonetic string. It not only takes overt speech from the speaker but also from other interlocutors, ie when the speaker we are looking at is in fact a hearer. The phonetic string is the counterpart of the phonetic plan. Like internal speech it serves as input to the speech comprehension system. The speech comprehension system analyses the phonetic string and the internal speech phonologically and grammatically into parsed speech, which is the input to the monitoring component of the conceptualiser. Listening to one’s own speech, be it internal or overt, is the prerequisite for self-monitoring.

In Levelt (1999) the surface structure is the interface posing the major rift in the language production system, ie the tripartite architecture becomes a bipartite one. However, all in all this modified architecture contains five modules of which the first one still is the conceptualiser, which has the same functionality as in the 1985 model.
2.3 Levelt’s tripartite architecture

So, in Levelt’s model production and comprehension of natural language take place in different components. The only shared components are the conceptualiser and the lexicon, and the differences of the tasks involved in production and comprehension corroborate this view, cf. the discussion of [McDonald 1987] on page 26. However, there are also proposals for bi-directional or reversible grammars, eg Performance Grammar [Kempen forthcoming] cf. also [Reiter & Dale 2000] 194–196 for a general overview. Such grammars can be used for production and comprehension alike. Results from empirical studies also indicate that at least from some point onwards the production and the comprehension system come together. For example, the results reported by [Levelt, Roelofs, & Meyer 1999] indicate that the lemma level[^1] is this point (p 7, assumption 3). Thus, the point probably is located within the formulator, not directly below the conceptualiser. The important point for conceptualisation is that the conceptualiser should not be seen exclusively as a device of language production but also of language comprehension. For this reason and because a lot research exists on self-monitoring – probably even more than on message generation –, I will give an outline of how INC can be extended by a self-monitoring component in chapter[^15].

Both knowledge representations in figure 2.2 are special views on human knowledge, focussing on which knowledge is available for the production of language. Yet, they are static views on representations. When looking at the system under a performance perspective there are three memory structures that determine the performance of the system. The first is working memory, where parsed speech and pre-verbal messages are stored. However, working memory is also used for other tasks, eg the selection and linearisation of utterances. Otherwise it would not be possible to plan multiple utterances in advance, eg it would not be possible to structure a small speech according to the rules of rhetoric in advance. Limitations of working memory influence which utterances are generated. If, for example, its capacity is exceeded utterances that were scheduled for verbalisation may be forgotten, and they are either not verbalised at all, or they must be planned anew. INC possesses a parameter that corresponds to how much working memory is available for conceptualisation. Varying it leads to different verbalisations.[^1] The other two memory structures are specialised buffers: the syntactic buffer stores the surface structure and the articulatory buffer the phonetic plan. Both buffers are limited in size, ie, as in the case of working memory, errors arise when their capacity is exceeded.

This means, preverbal messages as well as surface structures are never completely available at a given point in time – except for short utterances. However, the incremental mode of operation makes it (theoretically) possible to produce infinite utterances with limited resources, in particular storage capacity, which is not possible in a non-incremental mode of operation. Buffers are usually needed for

[^1]: A lemma is that part of a lexical item that contains the non-phonological information, cf. [Levelt 1989] 18iff).
[^1]: Resource limitations are one of the main topics of chapter 7. How different values for parameters influence the behaviour of INC is the topic of chapter 14. The parameter I am referring to here is the length of traverse buffer (LOTB).
incremental processing, because they serve to compensate differences in processing speed of the single components. Additionally, they allow a reordering of increments, which is required because of the rapid production of output. In chapter 5 the role of buffers in incremental processing will be elaborated in detail.

2.4 Modularity of conceptualisation

A central question to every model of a cognitive faculty is the issue of its modularity. Yet, since the issues of faculties and modularity themselves are not my topic, this is not the place to retell the story of the modularity debate, viz the debate over whether modules are suited to explain aspects of human cognition in the first place. What is more, this debate is still not settled. There are two particularly prominent approaches to cognition that totally reject the idea of modularity. The first is ‘cognitive linguistics’ (Lakoff 1987, Langacker 1987/1991/2000), which focuses on interrelations between different cognitive faculties instead of the interrelations within one cognitive faculty. Thus, there is no need for modularisation. The other approach is connectionism, also called parallel distributed processing (Rumelhart & McClelland 1986), which even rejects the idea of symbols and representations.

* More precisely, both approaches do not explicitly reject modularity. It is simply not regarded relevant. The upshot is that all elements used for one task can be considered belonging to one module, e.g. the neurons performing phonological encoding can be considered the phonological module. Yet, since all neurons are connected by the same type of links, there is no interface to other modules, say the morphological module, and the distinction between modules is not defined by the builder of the model but emerges. Note I do not use this self-organising notion of module.

† The symbol grounding problem is the problem that symbol definitions become eventually circular, because symbols are defined in terms of other symbols (Harnad 1990). Symbols are not connected to extra-symbolic properties, they are not grounded. However, there are other ways of dealing with this problem than rejecting symbols right from the start as I will argue in section 4.2.
6. ‘shallow’ outputs
7. association with fixed neural architecture
8. characteristic and specific breakdown patterns
9. its ontogeny exhibits a characteristic pace and sequencing

Fodor argues for these criteria by investigating the properties of input systems. Since more research has been done on input systems for visual perception and language comprehension rather than on output systems, Fodor only argues for such input systems. Nevertheless, his criteria are generally taken in some form or another to judge whether something is a module or not. Apart from input and output systems Fodor argues for the existence of transducers, which translate the different kinds of input into neural signals, eg transducers that compute neural signals from light, and a central system, which is a non-modular, general-purpose system for inference, deduction, and reasoning. While the nine criteria above describe systems for controlled processing, the central system is the system in which executive control takes place, ie this is the place of conscious, volitional processes.

Is it possible that only some of the nine criteria are met? And does the violation of one criterion mean that a component is non-modular? The studies of the polyglot savant Christopher lead Smith and Tsimpli (1995, 1996) to develop a ‘picture of language [that] crucially involves aspects of the central system as well, so the bald alternative of “modular/non-modular” is simplistic, indeed false’ (1996:16). In agreement with Fodor himself (Fodor 1985) they regard the most important criteria for modularity to be informational encapsulation and cognitive impenetrability (Fodor’s limited central access to the mental representations). Their findings reveal that ‘some system could be cognitively impenetrable but not informationally encapsulated, whilst the reverse relation is impossible’ (1996:13). For this reason they develop the notion of quasi-modules: while these ‘have the domain specificity of modules, they are not informationally encapsulated and they exploit a non-perceptual vocabulary’ (1996:1).

This fits in well with Levelt’s reluctance to speak of modules. Instead, he prefers to call the parts of his model components. Levelt characterises components as relatively autonomous specialists that generate characteristic output from characteristic input (1989:13–20). Besides its characteristic input a component only has minimal access to other information and is influenced only minimally by other components. This makes input characteristic input. Characteristic output is output that serves as characteristic input to other components.

Levelt and Fodor agree in that they consider the central system as the least modular component, and the more peripheral a cognitive component is the more modular it is. With regard to the language production system this means that the closer a component is to overt speech, the more informationally encapsulated and the less cognitively penetrable it is. Consequently, the conceptualiser is the least modular component and the articulator the most modular.

To further identify the status of the components with regard to modularity let
us look at another telling property. While automatic processes can run in parallel (in different modules), the controlled processing in the central system requires attention and is, therefore, sequential (Levelt 1989: 20f). As already indicated, nC uses parallel processes to model the sub-tasks of conceptualisation. This allows for an adequate explanation of the tasks, because their functionality can be separated and characteristic input and output can be defined. However, as I will show in part C, they also show certain interdependencies, which raises some doubts whether they are indeed as independent of each other as automatic processes. The main reason for this is that they use a shared memory* on which they operate (see chapter 9). The shared memory brings with it two main advantages: it simplifies the model (it reduces processing and storage redundancies), and it allows indirect feedback, which can account for some phenomena that a strictly unidirectional model would have difficulties to explain. (See page 98 and term 5.17 on page 104 on indirect feedback.) Furthermore, the shared memory fulfils a function comparable to the working memory of unified cognitive architectures, cf chapter 7. This enhances the cognitive adequacy of nC.

So, this indicates that the conceptualiser-internal components have some properties of modules but lack others: due to the shared memory the components are not fully informationally encapsulated. Yet, this does not mean that they must be cognitively penetrable as well, which is exactly the claim for the existence of quasi-modules. Although I will continue to call them components in the following, this always corresponds to quasi-modules.†

2.5 Non-modular aspects: conceptualisation and other cognitive tasks

In the previous two sections I described Levelt’s language production model and discussed the idea of quasi-modules in order to provide criteria that help estimating the degree of modularity of the human language production faculty. In this section I will show in more detail that some aspects of the language production system, especially of the conceptualiser, are non-modular, while in the following section I will illustrate which aspects are modular.

Levelt (1989) regards speaking as an intentional activity. This does not mean that there is no non-intentional speaking – think, for example, of speaking in your sleep or an exclamation caused by surprise –, but that non-intentional speaking can be considered being comparatively rare. Usually, a speaker has a communicative intention he wants to convey. However, as the subtitle of his book (from intention to

* I am referring to nC’s current conceptual representation (CCR).
† As you may have already observed I also use process and component more or less synonymously. The reason for this is that the four components of nC are realised and implemented as processes. This makes modelling the conceptualiser easier, because modular processes are easier to handle than quasi-modular components. Although both are two sides of the same medal I usually speak of processes when I refer to nC and of components when I refer to the quasi-modules in the conceptualiser.
articulation) indicates: ‘Where intentions come from is not a concern of this book’ (p 59). Thus, although intentions are Levelt’s starting point, he refrains from giving an account of what intentions are and how they arise. What is more, he restricts his discussion to communicative intentions, intentions that underlie speech acts. He describes this with the phrase: ‘The mother of each speech act is a communicative intention’ (Levelt 1989: 108). In fact, he makes two further restrictions. Firstly, he only considers illocutionary intentions: communicative intentions that are realised by no means of communication other than speech acts. Secondly, each communicative intention must be recognisable by the hearer as just that. A speech act that serves a different purpose, eg to deceive the hearer, has no underlying communicative intention. This means, speakers do follow Grice’s cooperative principle. For brevity’s sake I will continue to call the starting points of language production communicative intentions.

The fact that Levelt restricts his account of speaking to a subset of communicative intentions is no accident, because although intentions are essential for characterising our mental life they are an evasive notion. Despite some thousand years of thinking there is no commonly accepted definition. This is probably due to the fact that intentions are the result of conscious, volitional processes, and consciousness is one of the most disputed terms in philosophy. Because of this problem there is no way to fully explain language production. Even if it was is fully understood one day how the processes described in section 2.3 simulate mental processes, this would not include the computation of the ‘initial spark’, the spark that is the result of a volitional, conscious decision and sets everything into motion. Note that with respect to the question of weak or strong AI, this is an agnostic view, ie I neither support nor reject the possibility of an artificial system having conscious thoughts, because I aim at simulating rather automatic processes, which show only some aspects of controlled processing. One could say, these processes take place after the spark came into being. Intentionality is a strong argument that language production cannot be modular in the strict sense, because it is accessible to conscious thought, ie accessible to controlled processing. It also corroborates the view that conceptualisation is the least modular component of language production, because while all stages of language production are accessible to conscious thought, it is more difficult the farther away from the central system a component is located.

While these prospects may seem somewhat discouraging, the situation is not that bad. Although the initial spark may well be beyond scientific discovery, one can concentrate on the fact that humans are always situated in an environment. So, as I already argued in chapter 1 there is a strong data-driven aspect to conceptualisation and language production, and my goal is to shed some light on these data-driven aspects. While I, too, will not find the answer how initial sparks arise, inNC may give some answers to the question of how what one might call sub-intentions* arise in a data-driven scenario. The overall purpose (intention) of inNC is to produce online

Sub-intention is a term that I use, because I cannot think of a better one. I certainly do not want to introduce a new kind of intention. You can imagine quotes around it each time I use it.
descriptions of events, i.e., a verbalisation describing an event is produced while it takes place. So, if the communicative intention is describe what happens, the sub-intentions are communicative intentions underlying speech acts realising this goal. In a terminology communicative intentions are verbalisation goals. Speaking seen this way is a goal-directed activity, and sub-intentions are sub-goals that must be solved in order to reach the overall goal.

The reduction on the data-driven aspects of conceptualisation highlights the need for incremental processing even stronger, because it limits the available cognitive resources. Firstly, time is limited, because a piece of input must be processed while the next one already enters the system. Otherwise the system would not keep up with the input stream like humans do. Secondly, the available memory in which the input elements are processed is limited. According to Levelt (1989) this processing takes place in working memory, and the limitations of working memory are perhaps the best known and best studied issue in cognitive psychology. Hence, in order to accomplish the required tasks with the available resources, a mode of processing has to be used that enables the system to cope with the resources that are available. Incrementality is just the way do this: piecemeal processing that takes place on multiple stages in parallel means that the system is capable of reacting to new input all the time, even while already producing language.

In particular working memory sets language production in relation to other cognitive faculties, because working memory is not exclusively available for language production. In a natural (non-laboratory) environment a speaker is almost always occupied with additional tasks requiring working memory. Interference experiments show that there are interactions of utterance planning with other cognitive tasks requiring working memory, while there is no interference of such tasks and automatic articulation (Baddeley 1986, Eysenck & Keane 1995, Smith 1999). In these experiments the articulatory buffer of the participants is filled by letting them articulate in a non-controlled, i.e., automatic, manner while other tasks have to be performed that involve working memory. Since working memory performance is not significantly affected, one can conclude that working memory is not needed for (automatic) articulation; articulation employs separate resources. Other planning tasks, however, also require working memory, and the resource must be shared. Hence, working memory is only partly available for conceptualisation.

A further point that contradicts the modularity of language production on the level of conceptualisation is the interactional character of speaking. This means

More precisely, the description is produced shortly afterwards, because an event must take place before it can be described. But see chapter on the generation and verbalisation of expectations.

† In an offline verbalisation condition the production of language is not data-driven but memory-driven, i.e., the content to be verbalised does not enter the system from a perceptual input channel but is retrieved from memory. However, the problems of limited cognitive resources are almost the same. Although the cognitive load is smaller – there is no pressure to process new input and less tasks have to share resources –, processing takes place incrementally in the offline condition as well. The reason is that resources like working memory are as limited as in the online condition, and that it is equally possible to produce infinite utterances.
that speaker and hearer usually exchange roles in rather regular turns, and, what
is more, they share knowledge, called the common ground. (More precisely, they
estimate what knowledge they share.) This is not only knowledge about the dia-
logue, ie knowledge about what has been said and what will be said, but also know-
ledge about the world in general and the interlocutors in particular. Apart from the
shared discourse knowledge the spatio-temporal context, ie the setting in which
the dialogue takes place, and the social status of speaker and hearer must be taken
into account. A speaker must consider these factors when making a contribution
to the discourse. To take just one example: generating deictic expressions would be
impossible if the speaker did not observe these hearer dependent factors. All of the
five kinds of deixis listed by Levelt (1989) – person, social, place, time, and discourse
deixis† – are sensitive to extra-linguistic factors. The conceptualiser must integrate
them so that they can be used in an utterance.

As already explained in chapter 3, though, I only implicitly consider a hearer,
ie language production is not seen as the participation in a dialogue but as the
 generation of a monologue.‡ One major advantage of the online descriptions of
events is that it allows to leave aside this complex. The resulting simplifications
make it easier – or perhaps at all possible – to cope with the open-endedness of
conceptualisation that is due to non-modular factors.

One final remark on the context in which language production takes place. A
speaker can obviously address more than one person at the same time. Since I do
not even use a detailed hearer representation, I simplify matters in assuming a set-
ting in which there is only one hearer and suppose that my proposal can be exten-
ded to groups of interlocutors. To sum up, I do not account for the influences of
the speaker’s knowledge about other interlocutors on his verbalisations.

2.6 Modular aspects: language specificity of conceptualisation

After pointing out that some aspects of conceptualisation are non-modular, I will
now show that there are aspects that set conceptualisation apart from other cognit-

* Person deixis means, for example, the use of different available pronouns; you or he mean different
persons, depending on the context. When employing social deixis a speaker acknowledges the social
context, eg in German and French speakers distinguish a polite form (Sie and vous, respectively)
from an informal one (du and tu). An example for place deixis is the distinction between there and
here found in many languages, which is no absolute distinction but a relative one that depends on
where the speaker is located. Time deixis enables the speaker to use relative terms when referring to
points in time, eg yesterday instead of 1 January 1996. Discourse deixis finally allows to refer to
previous bits of discourse, eg this in The sun is shining. This is nice.
† Considering deixis makes clear once again that communicative intentions cannot only be realised by
language but by other modalities as, in particular by accompanying gestures and/or facial
expressions. Think of deictic gestures supporting an utterance containing here or there by pointing
‘here’ or ‘there’ or an utterance containing I, you, or she by pointing at the person referred to.
‡ Of course, I assume that this monologue is listened to by a hearer; otherwise there would be no
reason to produce language in the first place. But the hearer is not modelled explicitly.
ive tasks. I do this by asking the question of how specific to language production its components and tasks are. Component and task are closely related notions: a component is a part of a model/system that performs a task, and, vice versa, a task is what a component executes. Both must be specific enough to have characteristic input and characteristic output. This raises two more detailed questions:

1. Are the components specific to language production or are they general cognitive components?
2. Are the tasks specific for language production or are they general cognitive tasks?

As discussed in section 2.3 the most likely answer to the first question is that from some level onwards the components are shared between the language production system and the language comprehension system. Yet, if the higher components are shared for language production and comprehension, they might also perform other cognitive tasks. As the observable interferences mentioned in section 2.3 show, this is indeed the case, because components needed for conceptualisation, eg working memory, are also needed for other cognitive tasks, eg other planning tasks.

Finding the answer to the second question is more difficult, because it depends largely on what is considered a task. For example, if it is linearisation (in this generality; also often referred to as serialisation) then one will find many places in the mind where the task occurs. In language production we find linearisation of speech acts in the conceptualiser, phrase and word linearisation in the formulator, and phonological linearisation in the articulator, to take just the most prominent examples. However, there is an abundance of examples from other areas of cognition: linearisation of actions (first open the fridge, then get the milk bottle), linearisation of finger movements (when playing the piano), linearisation of eye-movements (when investigating a painting), and so on. In this view, there is one linearisation task that is executed by one linearisation component. *

If this was correct, however, it must be possible to find cases of systematic interferences, ie cases in which one kind of linearisation interferes with another one. Although this is an extreme example due to the generality of the proposed task, I know of no such interferences in the case of the conceptualiser, ie cases where linearisation of utterances interferes with linearisation of phrases or phonemes. The same is true for the other tasks of conceptualisation. Since there are no observable interferences and because of the differences in the computations for ordering utterances versus doing the same for actions, eye-movements, finger-movements, or phonemes, I stipulate that a task is always specific to a component. Yet, this does not mean that there is always only one task that is executed by one component. We already saw that language production and language comprehension share at least

* Assuming that one task executed by multiple components is implausible. If there exist multiple components, then it is only reasonable that the executed tasks are adapted to the particular problems, eg to the linearisation of eye-movements vs the linearisation of phonemes, which adhere to different principles because of the different subject matter.
some components. In fact, I want to argue for the opposite, i.e., for the view that components can execute multiple tasks.

I do this by zooming in on the issue of the language specificity of conceptualiser (the component) and conceptualisation (the task). This is the most difficult case in language production, because the conceptualiser is the least modular component and conceptualisation the least modular task as we saw in the previous section. While one may find it plausible that linearisation of words and syllables are different tasks, performed by different components, let alone linearisation of muscle movement to hit piano keys, this is not so easy for conceptualisation. Language specificity can mean two things:

1. How specific is conceptualisation with respect to the production of language?
2. How specific is conceptualisation with respect to the production of a particular language?

The first question is how specific conceptualisation is with respect to the production of language in contrast to processing conceptual representations to perform other cognitive tasks, e.g., a conceptual representation for navigating in an environment. The second question is whether conceptualisation is different for, say, speaking Zulu from conceptualisation for speaking Latin. However, the overall question still is: Is there only one conceptualiser performing one conceptualisation task? The answer is perhaps and no. There may be only one conceptualiser but there is not only one conceptualisation task. But why?

Let us start with the second question. Von Stutterheim (1999) and Rieckmann (2000) show that conceptualisations differ even for speaking languages as closely related as German and English. They investigated how participants describe events presented to them on a monitor (videos in the first case, drawings of sketch maps in the second case). What they found was that descriptions differ not only with respect to the linguistic means with which concepts are expressed (e.g., German knows no progressive forms), they also differ with respect to the use of the underlying conceptual representation: English native speakers tend to focus on the event itself, German native speakers tend to focus on its result. For example, when participants verbalise a scene in which they see a video of a street with a bus stop on the other side, and a woman, who is pushing a pushchair while she crosses the street, then English participants tend to produce an utterance like (1) while German participants tend to produce one like (2).

(1) *A woman is crossing the street.*
(2) *Eine Frau geht zur Bushaltestelle.*

‘A woman is walking to the bus stop.’

English and German native speakers might also have different conceptual representations. However, since the conceptual representations are built up by a process (construction in t~c~) such differences must go hand in hand with differences in the tasks.
Evidence pointing in the same direction comes from Carroll (1997) and from Levinson (1996, 1997), who present results from a cross-cultural comparison on spatial conceptions. Agreeing with Slobin (1996), these authors propose a *thinking for speaking*, which I already discussed in chapter 3 while humans are speaking, they think in a particular way. So, thinking for English must be different from thinking for German to some extent, because different parts of the conceptual representation are highlighted in the above utterances. Furthermore, the preverbal message must contain different information in different languages, e.g., it does not need aspect information in German, but it does so in English.

But can one conceptualiser perform different conceptualisation tasks? Bilin-guals are a strong case in favour of this view, because they speak both languages as native speakers. So, at different points in time their conceptualiser performs different conceptualisation tasks.†

The phrase by Slobin offers a solution to the first question as well: conceptualisation (thinking) for speaking differs from conceptualisation for other cognitive tasks. The alternative view is that native speakers of German and English differ not only in the way they verbally describe how to get from A to B but also in the way they plan their actions to get from A to B, which seems too far fetched. It cannot be fully rejected, because the importance of language for the way humans think is widely acknowledged. Even though the extreme form of the Whorf hypothesis (a human can only think what he can talk about) is accepted no longer,‡ language is the primary way for humans to gain knowledge and has, therefore, great influence on how we think of the world. For example, Wiese (2003) presents results of a (preliminary) empirical study that indicate that there may indeed be effects of a speaker’s language on conceptualisation.

To sum up, language production, conceptualisation in particular, is not independent from other cognitive tasks. However, it is independent enough to be investigated in its own right. So, is there a one-to-one mapping between tasks and components? This depends on what is considered a task. If we take conceptualisation to be the task in question, we find interdependencies between conceptualising for speaking and conceptualising for other tasks. But there are also aspects that are particular to language. So, the most likely answer is that conceptualisation is a quasi-module: it is a component between the central system and informationally encapsulated modules that is partly cognitively penetrable.

---

* The place in nC where conceptualisation becomes language specific is pvm-generation. Here, it must be decided whether language-specific information, e.g., temporal or aspect information, have to be included in the preverbal message.
† If bilinguals also had two conceptualisers they should be able to produce output in both languages at the same time, say writing German and speaking English, which – to my knowledge – is not possible.
‡ See, for example, the not only instructive but also entertaining essay on *The great Eskimo vocabulary hoax* by Pullum (1991).
3

CONCEPTUALISATION

Conceptualisation is notoriously hard to investigate. The main reasons for this are presented in section 3.1 together with a method with which it can be done nevertheless. Based on this I will argue that conceptualisation is non-linguistic, albeit particular for language production, and define the term ‘conceptualisation’. While its endpoint is rather clear-cut (preverbal messages), its starting point(s) and internal workings are much harder to explore. Therefore, I will describe the point from where I start and characterise the tasks that are performed in conceptualisation in section 3.2. In my investigation I use a view on conceptualisation that is strongly data-driven, which means in cognitive terms that it is driven by perception. I will give a working differentiation between the two in section 3.3, knowing that it cannot be an unambiguously defined interface but rather is a transition area with influences in both directions. In the last section 3.4 I will give an outlook of what it means for conceptualisation to be processed in an incremental fashion.

3.1 Investigating conceptualisation

In the previous chapter I argued that conceptualisation for language production is a non-linguistic task but specific to language production. That is, it differs to some extent from conceptualisation for other cognitive tasks, and its output is produced in order to be encoded linguistically. Thus, the purpose of conceptualisation is to bridge the stretch between the language production components proper – formulator and articulator – and other cognitive faculties, eg long-term memory or perceptual abilities. Before I describe the sub-tasks of the overall task conceptualisation for language production in more detail in section 3.2 I first illustrate difficulties in its investigation and present a more detailed account of the method with which I do investigate it.

There is no generally accepted view of conceptualisation for language production to be found in the literature. As a consequence, issues of conceptualisation are often treated in theories of language. One of the major problems for the investigation of conceptualisation results from this situation, as Nuyts observes:
The critical point is that neglect of dimensions beyond the linguistic leads to a tendency to deal with everything observable in terms of linguistic structures or processes. The result is models which have an ‘overcapacity’, accounting for phenomena, hence featuring structures and mechanisms, which do not belong there. One simple but prototypical case is the specification of selection restrictions on the argument positions of predicates in the lexicon. The grammar does not need to ‘know’ that one must be animate in order to breathe, or what one drinks is normally liquid. Such issues are solely a matter of one’s knowledge of the world, to be dealt with in the conceptual system. No need to duplicate this information in the grammar, as the grammar will be informed by the conceptual system about which elements of the conceptual state of affairs are to be lexicalized for communication […] (Nuyts 2001: 8)

There are three main reasons for this situation. Firstly, there is no agreement among researchers with regard to what they mean when they use the term ‘conceptualisation’. Secondly, it is difficult to determine the degree to which conceptualisation is specific or non-specific to a task, eg language production, as we saw in chapter 2. Therefore, the separation of conceptualisation from the task – language in this case – is difficult. Thirdly, and this quite likely is the cause for all the confusion, conceptualisation is not directly observable, because its output never surfaces directly, but always is input to (an)other system(s), the formulator in the case of language production, cf section 2.3. While the first two issues are no unusual problems of scientific investigation in general, the last one makes studying conceptualisation particularly difficult. A way out of this situation is to compare different behavioural patterns in different surface modalities in order to find commonalities, eg between language and gestures. In this way one can gain insight into the mechanisms that lie beyond the properties of the cognitive systems dedicated to a particular behaviour. In other words one can infer properties of conceptualisation (Nuyts 2001: 12).

As already pointed out, I use a different method, namely focussing on the data-driven aspects of conceptualisation. This is a means to strictly control its input. In this way the output becomes comparable – at least to a certain extent (Tomlin 1997), for example, has used this method in a study where he investigated how attention influences the order in which phrases are produced. He showed animated scenes on a computer screen of two fish swimming towards one another. A small arrow flashing over one of the fish shortly before they meet drew the participants’ attention to one of them. Then, one fish opened its mouth, devoured the other, and swam out of the visible area. Tomlin found that the entity in the focus of attention – no matter whether it is the swallowing or the swallowed fish – is generally produced as the syntactic subject in the verbal descriptions of these scenes. His bold claim is that this is a direct explanation of subject selection in English that needs no intermediary pragmatic or textual level; the element in the focus of attention is verbalised as subject. I consider the claim to be bold, not because I doubt its correctness for this particular case, but because it is not clear at all how it fits into a more general theory of selection and linearisation of content to be verbalised. Additionally, this
3.1 INVESTIGATING CONCEPTUALISATION

is a study with English speaking participants only; a corresponding study for other languages may yield quite different results. However, Tomlin’s results point in the direction of a very plausible mechanism for the incremental generation of preverbal messages: starting from the element in the focus of attention the preverbal message is generated increment by increment, where the element in the focus of attention is the first increment of a preverbal message. For emphasising the incremental way in which preverbal messages are generated I will also call them incremental preverbal messages.

One should think of incremental preverbal messages as ‘sequences of well-formed propositional structures on a sub-propositional level’ (Guhe, Habel & Tappe 2000). This means that the proposition representing a preverbal message is not produced as a whole but piecemeal, i.e., in increments, and the increments themselves can be described by propositions.

Griffin used an ‘inverse’ experimental setting in which the participant’s attention was not guided – as in Tomlin’s setting – but free (Griffin 1998, Griffin & Bock 2000). Instead, she used an eyetracker to keep track of the participant’s attention. The underlying assumption of this line of research is that eye movement reflects attention. The participant’s task was to verbalise pictures on each of which an event took place. The events were transitive actions, i.e., there are always two partakers A and B, and A is doing something to B, e.g., a dog is chasing a postman. Griffin’s main interest was the systematic temporal linkage between eye movements and the contents of spoken utterances. She finds that eye movements and, therefore, attention anticipate the order in which the partakers are mentioned in the corresponding verbalisations. In a study by Bock, Irwin, Davidson, & Levelt (2003) a similar setting is used, and the observed results are in accordance with these findings (see also Bock, Erwin, & Davidson to appear).

Such settings allow to gain insight into conceptualisation by correlating the input with the generated utterances. For investigating the online description of events I use a similar setting, which is schematically shown in figure 3.1. Scenes are presented on a computer screen, and the participant is instructed: ‘Describe what you

* The element in the focus of attention is the focussed element introduced in chapter 5, and the head of traverse buffer of chapter 4. The decision of whether the element in the focus of attention is selected for verbalisation is elaborated in chapters 11 and 13. The idea of incremental preverbal messages was introduced in Guhe, Habel, & Tappe (2000) and is described in detail in Guhe (in print) and in section 13.3.
see! The verbalisations are recorded on tape and then transcribed for analysis. Two types of scenes are used in the studies: motion events and the drawing of sketch maps. In the studies with drawing of sketch maps participants are asked in a first study to draw a sketch map of a way known to them, eg the way from the campus of the informatics department to the main campus in Hamburg. This is recorded with a drawing tablet. In the following verbalisation study the recorded drawings – not the resulting sketch maps! – are presented to a different group of participants. In the studies with motion events synthetic movements are generated with a computer tool and then shown to participants.*

The advantage of both settings is that the data used to play the sketch maps or the motion events can be taken not only for presenting the scenes to human participants but also as input for \( \text{inc} \). – Modulo a perceptual pre-processing unit (PPU) that is necessary to get from spatio-temporal coordinates to perceived entities (simple concepts) that serve as the actual input to \( \text{inc} \), cf section 8.3 – Thus, \( \text{inc} \) reads the same input data that is used in the verbalisation studies. This makes it possible to compare the output generated by \( \text{inc} \) with the analyses of the human verbalisation data in order to evaluate the quality of the model and the implemented system, cf chapter 14. Note, however, that \( \text{inc} \) does not produce text or spoken output but preverbal messages, so that the comparison is no one-to-one mapping.

3.2 Conceptualisation tasks

Before I describe the tasks that conceptualisation for language production consists of in more detail, let me characterise what I consider to be the overall task:

**Term 3.1** Conceptualisation for language production. *Conceptualisation for language production is the cognitive process that builds up conceptual representations for speaking and produces preverbal structures out of these representations. The preverbal structures can be encoded linguistically by a subsequent component.*

Since this characterisation should also be applicable in other frameworks, I avoid the terminology of Levelt (1989). Hence, I use *preverbal structure* for what corresponds to preverbal messages in Levelt’s model, and *subsequent component* instead of *formulator*. The formulation includes the characterisation of conceptualisation for language production as being a mediator between preverbal structures and the central system and/or the systems working in other modalities. Put differently, conceptualisation covers the stretch between linguistic encoding and other cognitive activities: reasoning about the world, spatial cognition, social cognition, *

---

* All studies were carried out with the help of tools that were developed in the project ConcEv. The programs *Zeichnung* (Turhan & Erichsen 1998) and *VirtualDraw* (Adiwidjaja & Gerhard 2000) allow to record sketch maps and to replay them on a computer screen. *ConMotion* makes possible to define movements of objects, eg taxiing planes, and play them to participants.
3.2 Conceptualisation Tasks

visual cognition, etc. In this view conceptualisation is specific for language production in both senses discussed in section 2.6 because it is for speaking, or, to use Slobin’s (1996) formulation again, it is thinking for speaking. From now on, I always mean this notion of conceptualisation for language production when I simply talk of conceptualisation.

The characterisation is given by using a close interdependence of conceptual representations and the conceptual tasks (processes) that operate on these representations. However, I will not ask the question of what is first, representations or tasks and then develop the one out of the other, because – as is well known – representations and processes can be converted into each other (almost) at pleasure. Instead, I try to approximate both by taking the available evidence for both and combining them.

Let us now turn to the description of the sub-tasks of conceptualisation already mentioned in section 2.3: construction, selection, linearisation, and the generation of preverbal messages.

Construction. Conceptualisation starts with the construction of conceptual representations. For the task of online descriptions of events this means that a stream of input increments is read in, and from the input increments a hierarchical representation of the external states of affairs is built up. I already demonstrated the need of hierarchies for representing events in the introductory example: a docking event consists of sub-events. While one could think of a mode of presentation of such an event in which the docking is perceived and conceptualised holistically, for instance, by speeding up the movements, normally an observer sees the sub-events and only then constructs the more complex docking concept from them. The crucial point is that event structures are hierarchical, cf Habel & Tappe (1999).†

Since conceptualisation is concerned with the processing of concepts, simple concepts have to be generated by a unit or units preceding the conceptualiser. As already mentioned, in INC such a preceding components is called pre-processing unit (PPU), cf section 8.5. For both domains (drawing of sketch maps and motion events) exists one PPU. A PPU reads in the information that is also used for the verbalisation studies and generates output that is the input to INC.‡ The elements that constitute the interface between PPU and INC are called perceived entities (PE),

* However, even if the docking is perceived and conceptualised holistically, then it is still possible to segment it into sub-events. Taking this possibility into account is one of the reasons that the concepts that are the input to construction are now called perceived entities and no longer basic entities as in Guhe, Habel, & Tappe (2000). The other reason is that basic entities is easily confused with basic level entities.

† In Habel & Tappe (1999), however, construction corresponds to segmentation/grouping and structuring, which are the tasks that take the perceptual input stream and segment or group its elements, respectively, and build up a hierarchical conceptual representation out of these elements. Construction encompasses more than this, cf chapter 10.

‡ The important aspect of where input comes from and that the structure of this knowledge depends on perceptual factors does not occur in the Levelt model, which simply says that the conceptualiser has access to different kinds of knowledge.
Having perceived entities as input to conceptualisation just says that the entities are perceived but not at which level of granularity they are, eg a square in the drawing sketch maps domain may constitute one PE as well as succession of PEs that represent the lines the square consists of. Therefore, construction not only has to perform groupings but also segmentations of perceived entities cf Habel & Tappe (1999).

The main part of constructing a conceptual representation is the matching of perceived entities onto conceptual knowledge available in long-term memory. In most cases this means that more complex elements are inserted into the conceptual representation that group together PEs. In this way hierarchical event structures are built up. If a concept available in long-term memory is matched only partially and if there is a possibility that the missing parts will arrive as the next PEs, an expectation can be generated. In the docking scenario one example of such an expectation is that the walkway stops moving when reaching the plane. Of course, it is an expectation only as long as the stop event has not occurred. Afterwards, the stop event is simply a regular event of the conceptual representation.

The building up of hierarchical conceptual representations is a complex task as we will see in chapter 10. In models of the conceptualiser that not only accept input of the kind described above but also verbal input from other interlocutors construction involves drawing inferences and making deductions such as presuppositions and implicatures (Grice 1975, Levinson 1983).

Selection. The next step towards the generation of preverbal messages is selecting events from the hierarchical conceptual representation for verbalisation. This is the task of generating sub-intentions discussed in section 2.5 corresponding to the first part of macroplanning in Levelt's (1989) model (macroplanning 1). For example, selection is the task that decides whether the sub-events of the docking event are verbalised, resulting in a four-utterance verbalisation of the scene as it is given on page 4, only one utterance like (1), or perhaps a mixture of both like (2).

1) Ein Flugzeug dockt an.
   A plane is docking.

2) a. Ein Flugzeug bewegt sich auf ein Gate zu.
   A plane is moving towards a gate.

* For this reason, they are not called perceptual entities, which might imply that the perceptual system operates on a fixed level of granularity, eg the level with the simplest concepts.
† A human will immediately and invariably say that it stops moving in order not to damage the plane. INC makes no such sophisticated inferences. The expectation is generated, simply because this stop is part of the overall docking event.
‡ See Menzel (1994) and Poth, Menzel, Pop, & Schröder (2000a) on the advantage of using expectations in incremental language processing.
§ As I already pointed out several times now, I am concerned here with the conceptualisation and verbalisation of events. Therefore, the selection task only considers events for verbalisation. Nevertheless, all tasks I describe here could be extended for processing other entities without changing the overall design and the subdivision into sub-tasks of the conceptualiser.
b. *Es stoppt beim Gate.*
   ‘It stops at the gate.’

c. *Es dockt an.*
   ‘It is docking.’

For the online description of events the selection task can be reduced to looking at each change that the construction task makes in the conceptual representation. This is possible because of the data-driven mode of operation. If the conceptualiser worked memory-driven the selection task could be used in two different ways. In the first the construction process does not receive elements from a perceptual preprocessing unit but from long-term memory instead; selection still would consider only events for verbalisation. The other requires a structure of the conceptualiser different from the one proposed here. In this view retrieving elements from long-term memory is the selection task proper, ie only those elements are transferred into working memory (the conceptual representation) that are actually verbalised. Although I sympathise with the first view, because it retains the separation of the construction of a conceptual representation and the selection of elements for verbalisation, the latter may still be the better solution due to its greater compactness. Separating the tasks makes the conceptualiser more flexible, because during the construction of the conceptual representation inferences, which are, for example, necessary for building up hierarchical structures, can be drawn. Conflating this into one task is to mix up two rather different tasks.

The selection task decides whether the element that was changed by the construction task will be verbalised. Each time a decision is made to utter a concept, a communicative intention (a sub-intention) has been generated. In a full account of the selection task a lot of factors must be considered in order to determine the appropriateness of the selected content, eg hearer model, discourse model, further communicative intentions that have already been decided upon but are not yet realised, social appropriateness, estimation of reliability (too speculative information is not verbalised), etc. Due to the neglect of a hearer model and the data-drivenness, two selection strategies have proved particularly successful. The first can be phrased as: *always take the most complex concept that is currently not selected for verbalisation.* In this selection strategy each event that is considered by selection is evaluated whether it has already been selected for verbalisation. If this is the case then there is no need select it a second time. For example, if the conceptualiser already decided upon generating an utterance describing an event that would result in an utterance like (2a) then there is no need to decide upon a second utterance describing the same event. The other part of this selection strategy is that the most complex event available is selected. For example, if the event described by utterance (3b) is selected for verbalisation then there is no need to select sub-events like (2a). Conversely, if (2a) is selected but not yet verbalised when the overall docking event is inserted into the conceptual representation then this simpler event is replaced by the more complex one so that only an utterance like (1) is generated. This strategy was developed on the basis of results from empirical studies. An evaluation on the
appropriateness of this selection strategy will be given in chapter 14.

The other selection strategy used by INC can be phrased as: retain the level of granularity if possible. This strategy selects an event only if another event of the same level of granularity has already been selected (but not yet verbalised). Only if nothing else is currently selected for verbalisation, events of another level of granularity are selected.† Since understanding the exact workings of this selection strategy require a more detailed knowledge of INC I postpone its full explanation. With this strategy sequences of utterances like the one given in the introduction on page 4 can be generated. Both selection strategies are described in detail in chapter 11.

In the general case, one communicative intention does not necessarily lead to exactly one utterance. Instead, one communicative intention may lead to multiple utterances, or multiple communicative intentions may be aggregated to one utterance. However, for most purposes at hand this is of no importance, for which reason I assume such a one-to-one correspondence − being aware that this is just a working hypothesis. Analogously, utterances are different from sentences. However, the formulator must treat the problem of how utterances are matched only sentences, because sentence is a grammatical notion. Thus, I use the term utterance, which is the linguistic correspondent to a communicative intention and which leaves open where exactly a sentence ends or where a new one begins.

**Linearisation.** Linearisation is the task of bringing the selected events (the sub-intentions) into an order that is appropriate with respect to the goal of the discourse. Although linearisation is perhaps the most investigated task of conceptualisation (see, for example, [Levelt 1983, 1989]), it plays only a minor role here. Firstly, linearisation is the most dispensable task of the four: in contrast to linearisation, verbalisation cannot work without construction, selection, and the generation of preverbal messages. Secondly, the online description of events provides a strictly chronological order of the input information, so that a reordering of events is expendable up to a certain degree. However, the difference of meaning in the example given by [Levelt 1989:138f]:

(3) She married and became pregnant.

(4) She became pregnant and married.

makes clear the need for a linearisation component. The two events in these examples (marrying and becoming pregnant) are verbalised in different orderings that correspond to the chronological order in which they took place. Deviations from the chronological order in the verbalisation must be marked:

* This selection strategy conforms to the Gricean maxims be brief, a sub-maxim of the maxim of manner, and do not make your contribution more informative than is required, a sub-maxim of the maxim of quantity ([Grice 1975]).

† This is motivated by the principle that an online description of events should in the ideal case consist of utterances that are generated at a constant rate. Reasons are that the (imagined) hearer should be kept up-to-date about what is happening, or that gaps in verbalisations are undesirable.
(5) She married after she became pregnant.

However, the principle of natural order proposed by Levelt (1989: 138, arrange information for expression according to the natural ordering of its content) is by far too simple. He develops this principle on the basis of a verbalisation task where he investigates how participants linearise static constellations of objects (Levelt 1983, 1989). In this task constellations of coloured dots linked by lines have to be described. The nature of this task may be the reason for postulating such a simple, general principle. Habel & Tappe (1999) show that the order of utterances is the result of far more complex linearisation processes.

Linearisation in the online description of events can be investigated by an extension of the task in which the participants have to verbalise concurrent events. This means that multiple events happen simultaneously. Since language is purely sequential the events have to be brought into a linear order.

Generation of preverbal messages. After the events to be verbalised have been selected and linearised, ie sub-intentions have been generated, an appropriate utterance expressing the speech act is produced. According to Levelt (1989), who calls this stage microplanning, there are four major aspects in the generation of preverbal messages: computing the accessibility status of the referents, topicalisation, propositionalisation, and acknowledging language specific requirements.

The accessibility status determines whether a referent is known and by which expression it can be referred to. The accessibility of a referent – and information in general – strongly depends on the state of the system: which entity has been used last, or, for our data-driven setting, which entity has been perceived last. Hence, questions of accessibility can be treated very elegantly by an approach using incremental processing. Topicalisation is the assignment of the topic role to one of the referents of a preverbal message. Propositionalisation is the task of generating a proposition for the sub-intention (event, speech act) to be verbalised. This proposition is the preverbal message expressing the event. It is the latest point where information

I use concurrent in two different meanings. The one that is used here is the verbalisation of concurrent events, the other the simulated parallelism in tNC. The parallel processes of tNC are (usually) not executed on a multiprocessor computer, which would allow true parallelism, but on a single processor computer, which requires the parallelism to be simulated.

The accessibility status of a referent is one of the following (in increasing specificity): inaccessible, accessible, in discourse model, or in focus. Inaccessible means that the referent has not been introduced and is not inferable by the hearer. Accessible means the same but the referent is inferable. For example, in preverbal messages resulting in the utterances,

(i) My favourite football team is Arsenal.
(ii) The goalkeeper is a genius.

Inferring the referent my favourite football team is neither introduced nor inferable from what has been said so far. Thus, it is inaccessible. The goalkeeper is introduced neither but accessible, because it is inferable by the fact that each football team has a goalkeeper. After these utterances Arsenal and its goalkeeper are both introduced into the discourse and can be referred to easier from now on. The former is in discourse model, the latter in focus.
that is represented in another than propositional format must be translated. The ability of referential nets to represent multimodal knowledge facilitates the translation between formats, cf section 4.3. A major aspect of propositionalisation is the assignment of perspective, or perspectivisation (Levelt 1989: 152–157). Habel & Tappe (1999) even only consider perspectivisation at this stage of conceptualisation. However, conceptualisation is not finished with perspectivisation: no preverbal message has been generated yet. The reason is that macroplanning is only concerned with the selection and linearisation of speech acts. This includes no decision on how a speech act will be realised, eg whether it is better for the intended purpose to talk of a particular person as Sarah or as David’s wife. I will take up the issues of perspectivisation and topicalisation again in section 5.4. Finally, the generated preverbal message must meet the language specific requirements of the formulator in the sense that it must account for requirements of the particular language in which the preverbal message will be expressed. For example, if a language overtly marks tense this information must be present in the preverbal message, while it may be left out in non-tense marking languages. Although I agree generally with the need for language specificity to a certain degree, I do not treat this problem in the following, cf also section 2.6.

Since in the Levelt model there is no feedback from formulator to conceptualiser, it must be guaranteed that a preverbal message can indeed be linguistically encoded by the formulator. Otherwise the verbalisation is erroneous or the attempt fails completely. Since humans do make mistakes, this is no absolute requirement; yes, a perfect model should even account for these mistakes. By and large, though, the encodability must be given. A possibility to assure it is to check whether a preverbal message is encodable before it is handed on to the formulator. However, this contradicts the idea of incrementality, because it interrupts the flow of information from one stage to the next: in order to perform the check the preverbal message must be available completely beforehand. Therefore, I consider encodability to be a requirement a conceptualiser must fulfil without an explicit mechanism.

Another problem at this stage of language production is the verbalisation problem addressed, for example, by Bierwisch & Schreuder (1992). Apart from the problem that there is no general, simple one-to-one mapping of communicative intentions onto preverbal messages, there is also no one-to-one mapping of the parts of a preverbal message onto lexical items. While the first problem causes no difficulties in the data-driven task at hand (see above), the second problem must be solved by the formulator, not by the conceptualiser. The reason is that the preverbal message is a special kind of conceptual representation, not a linguistic representation.

During the generation of preverbal messages additional means of expressing the

* In the field of NLG the encodability problem corresponds to the problem of the generation gap (Meteer 1990, 1991). Although one can quite simply assure that only encodable pre-linguistic representations are generated by restricting the possible representations, this is no real solution, because it means that the system undergenerates, ie the full richness and diversity of natural language is not used. This severely limits the quality of the system.
† When I use the term lexical item I make no distinction between lemmas and lexemes.
content to be communicated can be chosen. Especially prosodic information like loudness, rhythm, or intonational contour must be produced by the conceptualiser to some extent, eg to realise a contrast between two parts of an utterance. Other means include gestures or facial expressions.

3.3 Conceptualisation and perception

Although it is widespread consensus that perception and conceptualisation are two different levels of cognitive processing, the two are certainly not independent of each other. Conceptualisation and perception interact in both ways, ie there is not only an information stream from perception to conceptualisation but also feedback in the opposite direction from conceptualisation to perception. For example, expectations can lead to a faster processing of input if the expected concepts are indeed perceived (facilitation) or to a slower processing of input if the expected concepts are not perceived (inhibition). An entire, very successful experimental paradigm in psychology relies on this effect under the name priming. Another well-known example of the influences of conceptualisation on perception from the field of vision are ambiguous pictures like the duck–rabbit (eg depicted in Pinker 1999: 293). The duck–rabbit is a picture that can be seen as showing either a duck or a rabbit. What is seen can partly be controlled by conceptual decisions.

The setting used here makes necessary some form of perceptual pre-processing. However, although the setting is strongly data-driven and the verbalised scenes are perceived visually, I only provide a very coarse, technical model of perception, because perception is not my main interest. All that I assume is that perceived entities (Pes) constitute the interface between perception and conceptualisation and that Pes are computed from sequences of spatio-temporal coordinates. I make no detailed claims on what Pes consist of and what they represent. The crucial point is that they can be used by the construction task to search long-term memory in order to build up a conceptual representation.

Additionally, since I focus on the conceptualisation of events, perception and conceptualisation of other entities is only carried out to a degree of detail that is required for this purpose. In order to generate Pes the PPU, in which the perceptual pre-processing is done, segments the input according to the empirically founded cut-hypothesis by Avrahami & Kareev (1994: 239): 'A sub-sequence of stimuli is cut out of a sequence to become a cognitive entity if it has been experienced many times in different contexts'. Since nC is capable of generating expectations, it is principally also capable of generating feedback to the PPU in order to influence perception. For example, if nC generated an expectation then it can send this information to

* Since these information streams consist of increments, they are called increment streams, cf page 52.
† Expectations can be seen as the conceptual counterpart of a pre-activation in the perceptual system.
‡ Priming effects do not only reveal influences of conceptualisation on perception but are equally used to illuminate the internal workings of the perceptual system.
§ Technical means that the perceptual pre-processing unit (PPU) is adapted to the needs of nC.
the PPU so that the PPU may easier recognise the next input. In case the next input does not match the expectation one would get the also empirically observable behaviour that recognition is inhibited. Another example is that if a PE is not exact or detailed enough the PPU can send a request to the PE to provide better/more information about the PE. However, up to now these techniques are not used.

3.4 Incremental conceptualisation: an outlook

Although incremental processing is the topic of part B I want to demonstrate in this section that increamentality is not just one of multiple possibilities to perform conceptualisation but one of its essential properties. I will do this by elaborating three points: the limitation of cognitive resources, the influence of the ordering of increments, and the temporal interleaving of the sub-tasks.

Incremental processing can reduce the complexity of computations. Therefore, it is suited to cope with the limitations of cognitive resources, in particular the limitations of memory and time. An idea closely related to incrementality is anytime processing. In anytime processing a result can be requested at any time, but the quality of the result improves over the course of time. Put differently, the depth of processing depends on the availability of the resource time. Since cognitive resources are limited – think, for example, of the limitations of working memory – and conceptualisation requires resources, the limitations must be overcome, and a way to do this is incremental processing.

However, incremental processing also means that the optimal result with respect to all conditions may not be found. For instance, Dale & Reiter (1995), on the basis of work by Pechmann (1984), provide an incremental algorithm for the generation of referring expressions. This algorithm generates a uniquely identifying referring expression for an object that is chosen for verbalisation from a set of objects. It does so by incrementally selecting properties of the object that distinguish it from the other objects in the set. The performance of the algorithm is not optimal in terms of number of properties that are selected for describing the object. For example, if the set contains three objects: a small, white bird, a small, black bird, and a big, black bird, the algorithm may generate the big, black bird, although the big bird would be the optimal description in the sense of a minimal description. The algorithm is optimal†, though, with respect to run-time (efficiency): since it needs no backtracking it runs in linear time depending on the number of distractors, viz the objects that are not verbalised. Yet, the results generated by this algorithm reflect the performance of human participants. Apart from this example there is more, overwhelming evidence that language is produced and processed in

* More precisely, a result can be obtained for any amount of time spent on a computation, and the quality of the result improves the larger the available amount of time. I will compare incrementality and anytime processing in detail in section 7.3

† Optimal not in the strict sense. There may still be an equivalent algorithm that is faster.
3.4 INCREMENTAL CONCEPTUALISATION: AN OUTLOOK

an incremental fashion by humans, for example, De Smedt & Kempen (1987), Kempen & Hoenkamp (1983, 1987), De Smedt (1990a, b), Kempen (forthcoming), Kempen & Harbusch (in press), Altman & Kamide (1999), Chater, Pickering, & Milward (1995). Altman & Kamide (1999). Thus, incremental processing can not only be used to save resources: since humans produce language incrementally they do not always produce an optimal verbalisation. Hence, a model using incremental processing can account for such phenomena.

In addition to saving resources incremental processing can also be used to leave certain information implicit. That is, the sequence of the increments of a preverbal message can encode information for the formulator, cf section 1.6. Output generated by formulators like the one of Kempen & Harbusch (in print) depends on the sequence in which the input increments arrive. A reordering or temporal suspension of operation is used only if otherwise the grammaticality would be violated. For example, topicalisation and perspectivisation do not need to be explicit transformational processes (like, for instance, in Ziesche 1997), but are side-effects of incrementality. The incremental generation of a preverbal message commences with one concept, ie this concept is handed on as first increment of a preverbal message to the formulator. This is the referent the preverbal message is about in Levelt’s formulation (Levelt 1989: 151) and, therefore, the concept that is to be realised as topic. However, since I cannot present any empirical evidence for this claim, it is a working hypothesis. Nevertheless, the model by Kempen & Harbusch (in print) shows that this method makes possible to build generators without a separate perspectivisation component.

Deciding upon the first increment of a preverbal message is at the same time the first part of perspectivisation. For example, the decision, which of the two following utterances will be produced, is already made.

(6) David loves Sarah.
(7) Sarah is loved by David.

If the concept David is generated before Sarah as increment of the preverbal message (6) is realised; if the order is inverse it is (7). The two utterances describe the same state of affairs but from different perspectives: in the first it is an statement about David, in the second one about Sarah, cf also section 1.6.

The third of the issues mentioned above, the temporal interleaving of the con-

* In INC this depends on the interplay of selection and PVM-generation. Since concepts are represented by refO’s the first refO that is handed on to the formulator ‘sets the stage’. All following refO’s depend on this first one. More generally, later refO’s always depend on the ones generated earlier.
† A second step in taking perspective – not related to the order of increments – is the decision of how to refer to an entity. For instance, Sarah cannot only be referred to with her name (Sarah) but with many different descriptions: Anna’s daughter, the mother of Peter’s children, Peter’s wife, etc. The algorithm generating these expressions basically consists of deciding upon a designation associated with a refO and then recursively finding further designations for the refO referred to by this designation in order to ground it, cf chapter 13. The algorithm is resemblant to Dale & Reiter (1993).
ceptualisation tasks, has important consequences for the way conceptualisation proceeds. Consider the point of time when the generation of an incremental preverbal message commences. Obviously, the preverbal message that is generated can only contain information that is present in the conceptual representation. Thus, a preverbal message describing an event can differ from a preverbal message that is generated later even if it describes the same event. This is particularly true if the event was not completed before the generation started but can also be true afterwards, e.g. if additional inferences about the event were drawn in the meantime, e.g. connecting the event to other events.
CONCEPTUAL REPRESENTATIONS

Conceptual representations are a necessary prerequisite for conceptualisation. Therefore, after in the last two chapters I laid out the connections of language production and other cognitive tasks and described the tasks of conceptualisation, I will now elaborate my notion of concepts and what representations are used for conceptualisation. In the first section I will start by discussing the building blocks of conceptual representations, concepts. A hard problem for approaches using concepts is the symbol grounding problem, for which data-driven settings offer a solution. I will shortly comment on this in section 4.2. In 4.3 I will give a more detailed account of the representational formalism referential nets that is used throughout this investigation, and in 4.4 I will point out that the representations used in conceptualisation are always internal representations of external states of affairs. This means the representations are particular to an individual human or an individual system, and, therefore, they are subjective and not objective. How to represent events will be the topic of section 4.5 where I will also describe how a conceptual representation for the example of chapter 1 is built up incrementally. Finally, I will argue in section 4.6 that additional knowledge apart from a representation of the state of affairs is necessary for conceptualisation, or, put differently, that thinking for speaking requires representations for speaking.

4.1 Concepts

Despite the fact that the notion concept is central to several scientific disciplines there is no agreement among scholars about a clear-cut definition. In this respect we face the same problems as with the term intention on page 177. The MIT Encyclopedia of the Cognitive Sciences [Wilson & Keil 1999] defines concepts as follows:

The elements from which propositional thought is constructed, thus providing a means of understanding the world, concepts are used to interpret our current experience by classifying it as being of a particular kind, and hence relating it to prior knowledge. The concept of 'concept' is central to many of the cognitive sciences. [Hampton 1999, p. 176]
Two points in this characterisation are important. Firstly, concepts serve as a means of classification, eg to identify different planes as being instances of the concept PLANE and different walkways as instances of the concept WALKWAY. Secondly, concepts are the means to relate new knowledge to prior or given knowledge, ie to relate the plane instance that was just perceived to the conceptual knowledge about PLANES in order to retrieve additional information about this entity. Since an elaborate discussion of the nature of concepts – what could be called the ontology or epistemology of concepts – is far beyond the scope and the aim of my investigation, I will leave it at that and now concentrate on which concepts are used and how they are used as building blocks of the conceptual representations used here.

Among the plethora of ways to classify concepts in conceptual representations three are particularly important for the purpose at hand:

1. part-of hierarchies
2. subsumption hierarchies (ontology, sortal hierarchy, taxonomy)
3. representations of entities (individuals) vs representations of categories

The necessity of hierarchical conceptual representations was already pointed out in chapter [1], eg in figure 1.1 on page 10. While part-of hierarchies are commonly accepted in the case of concepts representing concrete entities of the world, eg the fact that a PLANE consists of multiple parts, they are less common in the case of event representations. Nevertheless, they are required as we already saw when we looked at the example in chapter 3 where two simpler events were taken together and constituted a more complex one. Further reasons to treat event concepts similar to object concepts will be given in section 4.5.

In the following I will often talk of simple and complex concepts. Simple concepts are neither basic concepts nor basic level concepts. The reason for not calling them basic concepts is that I do not claim that the concepts I use here are in any way atomic or the most simple ones from which all other concepts are constructed. They are also no basic level concepts (Rosch, Mervis, Gray, Johnson, Boyes-Braem 1976), which I do not consider as such. Simple concepts are really just simple. In particular they are simple enough to be computed by the pre-processing unit (PPU). Thus, perceived entities (PES, section 3.2) are examples of simple concepts. Hence, simple concepts are not necessarily concepts that are at the bottom of the part-of hierarchy. (Nevertheless, this is the case in the representations throughout this investigation.) The grouping and segmentation operations of the construction task create concepts on a neighbouring level of the hierarchy – up or down, respectively. Thus, there are infinitely many levels of concepts.

Apart from distinguishing concepts by their position in the part-of hierarchy, one can also discriminate them by their sort. The major distinction in the domain of motion events is the one between the sorts situation, spatial_entity, and

* The computation of PES is not within the scope of this work. An outline of how they are computed from spatio-temporal input will be given in section 8.5

58
object as we saw in chapter. The sorts themselves are also hierarchically organised. They define the subsumption hierarchy. For example, the sort situation can be made more specific to state, process, or event; and these sorts can then be specified further, cf figure on page. The usual parlance is to say that an event is a (or isa) situation. Hence, this kind of hierarchy is sometimes also called isa-hierarchy. Sorts allow to subdivide a conceptual representation into multiple layers of representation, eg the situation and the object layer.

The third type of distinction of concepts listed above is important in the context of relating perceived entities that enter the system with prior knowledge. Establishing such relations is called categorisation. The MIT Encyclopedia of the Cognitive Sciences defines categorisation as follows:

In other words, categorisation is the task of finding a category for a perceived entity. I use the following notions of concept and category in the following:

**Term 4.1 Category.** A category is a set of abstract or concrete entities that share a set of common properties.

**Term 4.2 Concept.** A concept is the mental representation of a category or an entity. They are the symbolic elements from which a conceptual representation is constructed.

With this definition of concepts I deviate from the way they are normally used in psychology, where they are mental representations of categories only, but not of entities. For example, characterises concepts as follows: ‘mental representations of categories are the entities psychologists mean by concepts. […] a category usually refers to a group of objects in the world, whereas a concept refers to a mental representation of such a group’. See also on the issue of concepts in general and a current exposition from a psychological point of view.

The advantage of my definition is that it saves me unnecessary differentiations in the following. The main reason is that for my purpose the distinction of knowledge about a set of entities and a single entity plays no role. It simplifies, for example, to define processes that retrieve or modify knowledge about something perceived. That is, the same mechanisms retrieve knowledge about the categories and entities known to the system. Additionally, it allows to call by the same name the items stored in the nC’s concept storage (cs) and the ones in the current conceptual representation (ccr). Using concepts in this way blurs issues that are important in other respects, in particular the issue of abstracting over properties of different
individual entities. Yet, for the purpose at hand this causes no problems.

The view on categorisation I propose here is in agreement with Gärdenfors (2000), who takes ‘categorization to be a rule for classifying objects’. That is, categorisation is seen as a cognitive task, not an abstract relation between entities and categories. Note that categorisation processes of a different kind are required to build up (learn) concepts representing categories. A more detailed account of the approach taken by Gärdenfors follows in the next section.

Categorisation in the model proposed here is a two stage process. The first stage takes place in the 

The second stage follows in

1NC, which tries to summarise a set of pes to a more complex concept or to analyse a concept into simpler concepts. This is done recursively for all concepts generated in this fashion, ie also for the ones computed by 1NC. Each newly perceived entity (and each self-generated concept) is compared to the available knowledge about concepts. For example, 1NC can relate the pe representing the plane of the introductory example to knowledge about planes in general, which it retrieves from the cs, eg the fact that a plane has an owner. If the owner is not known an empty concept can be introduced into the conceptual representation. This concept stands for the owner and also represents that nothing more about the owner is known than the fact that he/she/it owns the plane. However, the more important function of category knowledge serves to group simpler concepts to more complex concepts (segment them into simpler concepts), eg the simpler events in the introductory example can be grouped to the docking event.

How can the category of an entity be determined? The main problem of categorisation was already mentioned: entities can be categorised in a limitless number of ways. Categorisation is mainly accomplished by measuring similarity. Two ways to do this are the feature-based and metric-based measurement of similarity (Smith 1995). In the former the similarity between two concepts is measured by counting the features they have in common in contrast to the number of features they differ in. In the latter, similarity is measured as metric distance in a (multidimensional) vector space or a metric space. There is a third approach to categorisation, described in the MIT Encyclopedia of the Cognitive Sciences as follows:

An alternative to the similarity view of categorization is that theories provide conceptual coherence [. . .]. The theory-based explanation of categorization is consistent with the idea that concepts are comprised of features or properties. [. . .] These explanations go beyond similarity models in arguing that underlying principles (often causal) determine which features are relevant and how they might be interrelated [. . .].

(Medin & Aguilar 1999: 104)

1NC employs a categorisation method that first uses theories in the above sense to determine concepts that are then compared by measuring their similarity. In the case of the grouping and segmentation of concepts the similarity of two concepts
is the ratio of shared and non-shared parts, cf section \[10.3\]. Thus, it is an approach more similar to the feature-based method. Note that concepts need not be identical in order to be grouped; missing parts can be expected to be perceived later on.

Alas, the measurement of similarity has the same ubiquitous problem as categorisation in general: all entities are similar in infinitely many dimensions. Hence, one has to decide with respect to which features/metrics similarity is computed. This is a general modelling problem, for which reason I will not pursue it as problem in its own right but will only hint at it when the similarity between the used concepts is discussed in section \[10.3\].

### 4.2 A remark on the symbol grounding problem

A pervasive problem for symbolic approaches to cognition is the symbol grounding problem, brought forward by [Harnad (1990)](#). It addresses the problem that symbols, in particular concepts, cannot be defined exclusively by other concepts, because that results in circular explanations. In order to avoid this circularity concepts must be expressed in non-conceptual terms, cf also the [Encyclopedia of the Cognitive Sciences](#) (Hampton 1999: 177). In the data-driven approach pursued here the problem still exists but is less grave. The reason for this is that because of the data-drivenness the used concepts are connected to and defined by extra-conceptual properties. This solves the problem at least for cases in which the simple concepts of a representation are tied to perception. A solution along these lines is also suggested by [Harnad](#).

Since a solution of the symbol grounding problem is not required for my purposes, I will not provide an elaborated solution. Nevertheless, the approach by [Gärdenfors (2000)](#) is similar to the one I use in \[1\]. I will therefore outline it in order to show how symbols (concepts) can be grounded.\(^\star\) Readers familiar with such solutions can safely skip the remainder of this section.

Gärdenfors’s account is based on the notion of dimensions. Examples of dimensions are temperature, pitch, or brightness, which are all tied tightly to the sensory systems. But there are abstract, non-sensory dimensions as well. He then defines a domain as ‘a set of integral dimensions that are separable from all other dimensions’ (p 26). The colour domain, for example, can be described by the dimensions hue, chromaticness, and brightness. Yet, there can be no temperature–colour domain or the like, because such dimensions would not be integral. Integral dimensions depend onto each other to a certain degree.

Based on dimensions, Gärdenfors defines the core notion of his approach, conceptual spaces. A conceptual space is a collection of one or more domains (p 26).

\(^\star\) Dörner (1999) proposes a solution based on similar ideas, although he does not address the symbol grounding problem directly.

\(^\dagger\) My approach is also close to the one by Gärdenfors in that he describes it as ‘instrumentalist cognitive epistemology’ (eg on page 106). This means that he is not interested in ontological and epistemic questions of concepts but only how they can be used.
Thus, all in all there are three levels. Located between the symbolic (conceptual) and the sub-conceptual is the level of conceptual spaces that connects the two. Conceptual spaces, or, to be more precise, region within conceptual spaces, are then defined as natural properties and natural concepts. Gärdenfors characterises these in the following way: ‘A natural property is a convex region of a domain in a conceptual space’ (p 71). ‘A natural concept is represented as a set of region in a number of domains together with an assignment of salience weights to the domains and information about how the region in different domains are correlated’ (p 105). Thus, properties and concepts differ in that a property is defined with respect to one domain, eg in the colour domain red is a property, and a concept is defined with respect to multiple domains, eg apple as in table 4.1. It shows the region for the domains in which the value for a particular apple can be located.

<table>
<thead>
<tr>
<th>DOMAIN</th>
<th>REGION</th>
</tr>
</thead>
<tbody>
<tr>
<td>colour</td>
<td>red–yellow–green</td>
</tr>
<tr>
<td>shape</td>
<td>roundish (cycloid)</td>
</tr>
<tr>
<td>texture</td>
<td>smooth</td>
</tr>
<tr>
<td>taste</td>
<td>regions of sweet and sour dimensions</td>
</tr>
<tr>
<td>fruit</td>
<td>specification of seed structure, flesh and peel type, etc according to principles of pomology</td>
</tr>
<tr>
<td>nutrition</td>
<td>values of sugar content, vitamins, fibres, etc</td>
</tr>
</tbody>
</table>

Table 4.1: Domains and regions for apple Gärdenfors 2000:103

The definition of natural concepts contains the additional notion of salience weight. Domains are weighed with regard to their respective salience. The salience weight depends on the context in which a concept is used: for eating an apple its taste is more dominant than for throwing it (p 103). Giving some domain particular attention corresponds to taking perspective when using the concept. Additionally, salience weights are also influenced by the knowledge and interest of the user.

The view of concepts proposed here stands in stark contrast to the atomistic view, put forward, for example, by Fodor 1975, 1998. According to this view, a concept is an unanalysable whole. However, it does not account for the complex conceptualisation tasks that have to be performed, especially the task of identifying concepts on the basis of perceptual stimuli. The atomistic view suffers from the problem that it makes no distinction between properties and (other) concepts like Gärdenfors does. Instead, both are simply concepts.

* Deacon 1997 proposes a similar tripartite distinction.
† Properties are regarded as a special, simple kind of concepts (p 60). Natural means that properties ‘are, in a sense, natural to our way of thinking’ (p 66).
4.3 Referential nets

The representations in this investigation are given in the formalism referential nets (refNets), a way to represent knowledge that is particularly suited for the purpose at hand. RefNets consist of interrelated referential objects (refOs) representing entities [Habel 1982, 1986, Eschenbach 1988]. Formally, a refO is a term.

There are three main reasons for choosing refNets. Firstly, refOs are the major means of structuring knowledge, which could be called an object-oriented way to represent predicate logic. Consequently, the representations focus on the knowledge about individual concepts. The conception of refNets was developed in order to model the structure and organisation of (human) memory and the way it supports the processes of storing and remembering [Habel 1986: 11ff]. Secondly, refNets are especially suited to connect the different levels of representation used here. That is, it is possible to represent semantic and conceptual knowledge as well as to connect a refO to the perceptual information provided by the PPU. Additionally, refNets can integrate multimodal knowledge, e.g., pictorial and gestural knowledge, cf chapter [Habel 1986: 59]. Thirdly, refNets facilitate incremental processing, which I will elaborate at the end of this section.

Let us go into the technical details of the formalism. Consider the following part of a refNet, which might be used in the conceptual representation of the introductory example:

\[
\begin{array}{c}
\text{plane} \quad r_1 \quad 'CK-314' \\
\text{owner('LUFTHANSA')} \\
\eta x \text{plane}(x)
\end{array}
\]

In this notation, \( r_1 \) is the refO term. The lines leading towards \( r_1 \) connect it to the expressions on either side. The ones to the left are attributes, the ones to the right designations. The basic inventory of which expressions are constructed consists of three sets: variables (VAR) and operators (OP), and operator effects (EFF := \{t, f\}) [Habel 1986: 59]. The effect of an operator shows whether the result of its application is a term (t) or a formula (f). Operators are specified by the function \( \text{TYP} \):

\[
\text{TYP} = \text{OP} \rightarrow \text{EFF} \times \text{N} \times \text{N} \times \text{N}
\]

The three \( \text{N} \) values stand for the number of variables bound by \( \text{OP} \), and the terms and formulas \( \text{OP} \) has as arguments, respectively. I will use the shorthand notation

\[
\text{TYP}(\text{op}) = (e, m, n, p)
\]

in the following in which \( e \) determines the effect. This notion of types is also used in the semantics of programming languages and logic, which goes back to [Kalish & Montague 1964].

Expressions are sorted [Habel 1986: 66], i.e., the set \( \text{SORT} \) is a set of names of sorts, and each expression has a sort frame. The sort frame defines of which sort the
arguments of an expression must be in order for the expression to be sort-correct, eg the expression chpos (change of position), which represents the movement of an object along a path in a situation (see also section 4.5), has the sort frame (situation, object, path). However, sort-incorrect expression are not excluded by the formalism (Habel 1986: 66).

This basic inventory is used to form different kinds of expressions. Apart from refOs, which are of type ⟨t, o, o, o⟩, especially two kinds of terms play an important role: names, which have type ⟨t, o, o, o⟩, and descriptions, with type ⟨t, m, n, p⟩, with n > o or p > o (Habel 1986: 117). Examples of names are ‘DAVID’, the proper name of a person, or ‘CK-314’, the flight number of the plane in the introductory example. Names are written in capital letters and quotation marks.

Descriptions are either functional expressions, or they are constructed with a description operator. An example of the former is owner(‘LUFTANSA’), which states that the represented entity is owned by Lufthansa. Two points are important here. Firstly, the owner relation expressed by the functional expression can refer to another refO, eg owner(r2). Then, the owner of the entity is represented by r2, which has the advantage that additional knowledge about the owner can be represented. Secondly, since functional expressions are functions, they must uniquely refer to an entity, ie they must constitute a many-to-one or a one-to-one relation. In the case of ownership one can assume an entity only has one owner. It must be determined for each representation individually what is to be represented as functional expressions, because this depends on the content of the representation.

Descriptions that are constructed with a description operator are of the form: op var formula with the operators op ∈ {ι, η, all_t, some_t}, the variables var ∈ {x, y, z, . . .}, and formula being a formula of predicate-logic. The operators reflect the cardinality of the refO and the definiteness of the designation (Habel 1986: 137), cf table 1.1 repeated here as table 4.2. For example, ηx plane(x) stands for a plane and all_t x plane(x) for all planes.†

<table>
<thead>
<tr>
<th></th>
<th>DEFINITE</th>
<th>INDEFINITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cardinality = 1</td>
<td>ι</td>
<td>η</td>
</tr>
<tr>
<td>cardinality &gt; 1</td>
<td>some_t</td>
<td>all_t</td>
</tr>
</tbody>
</table>

Table 4.2: Operators in referential nets

Names and descriptions form the two sets NAM (names) and DESCR (descriptions). The latter contains the subset of closed descriptions DESCR.cl. As usual, closed terms (here: descriptions) are defined as those terms that contain no free variables. The union of NAM and DESCR.cl constitutes the set of designations (written to the right of the refO term):

* If there is a group of owners this can be represented by a refO standing for the group. The functional expression then refers to the refO. Yet, it is not possible to have multiple refOs representing owners.
† These operators construct terms, thus, they are of type ⟨t, m, n, p⟩, while the well-known quantifiers ∃ and ∀ construct formulas, ie they are of type ⟨f, m, n, p⟩.
4.3 REFERENTIAL NETS

(4) \( \text{DESIGN} = \text{NAM} \cup \text{DESCR.cl} \)

Since refOs are also defined as terms, terms are members of one of the following sets (Habel 1986: 117):

- referential objects (REFO)
- variables (VAR)
- names (NAM)
- descriptions (DESCR)

While designations represent linguistic knowledge – especially about meaning – the attributes of a refO* represent inferential knowledge. Examples are the information about the sort of a refO, which is obligatory and always stands in first position (\( \text{plane} \) in example (1)). If the sort of an entity is unknown the top-sort is used, cf figure 4.1 on page 69. Note that since the sort hierarchy is formally a lattice, it also contains a bottom sort. Temporal relations between situation refOs are also represented by attributes, eg before(\( r_3 \)) as attribute of \( r_2 \) represents the relation \( r_2 \) before \( r_3 \). Thus, attributes can have values, which can also be lists, written as \([ \ldots ]\). An example of an important list attribute is \( \text{parts} \).

Assume that \( r_5 \) represents Oscar Peterson, \( r_6 \) Ray Brown, and \( r_7 \) Herb Ellis. Then the refO representing the The Oscar Peterson Trio, say \( r_8 \), has the attribute \( \text{parts}(\{r_5, r_6, r_7\}) \).

Note that the sort \( \text{plane} \) and the description \( \eta_x \text{plane}(x) \) both encode the fact that \( r_1 \) is a plane. However, while the former stands for an essential property of the refO, from which, for example, it can be inferred that the entity is capable of flying, the latter says that it can be described as \( a \text{ plane} \) (Habel 1986: 156). This is the reason that mainly designations are used for the generation of preverbal messages.

As already stated, refNets are mainly used for representing propositional knowledge. Yet, they can also represent knowledge in other modalities, in particular spatial knowledge, for which the formalism is extended by depictions (Habel 1987, 1988). Similar extensions for other modalities are possible, which allows a translation from other modalities into the propositional one. The \( \text{perceived entities} \) from which the conceptual representation is constructed are identified by non-linguistic properties. Such properties as well as knowledge retrieved from long-term memory

* Designations can also have attributes. However, I will only use them once, cf section 4.3.
† In previous publications and in Eschenbach (1988) this attribute was called \( \text{sum} \). Yet, for events this is problematic, because what is regarded part of an event is flexible and not as well-defined as that a cube is the sum of its six sides. In the docking example the complex \( \text{docking} \) event may be constructed even if the event representing the start of the walkway movement was missing.
‡ The difference between attributes and designations is close albeit not identical to the difference of semantic memory (attributes) and episodic memory (designations), cf, for instance, Tulving (1999). Properties of the kind used by Gärdenfors (2000), ie properties tightly connected to perception, are represented mainly by attributes and only partly by designations. Properties connected to linguistic knowledge are represented mainly by designations and only partly by attributes. However, distinguishing what should be represented as attribute and what as designation cannot be done in general but must be decided for each case individually. Since a representation always is an abstraction, this depends strongly on what the representation is used for.
I want to conclude the discussion of referential nets with two general remarks and two remarks concerning the suitability of referential nets for incremental processing. The first general remark is that a refNet need not be consistent but allows to represent contradictions. This makes possible to adequately represent cognitive knowledge, because humans (can) have inconsistent representations. Thus, contradictions must be resolved by the processes operating on the representations.† The second general remark is that it is possible to represent default knowledge, eg the famous case that penguins are birds, birds can fly, but penguins cannot. Hence, the default knowledge is overwritten.

The first reason refNets are suited for incremental processing is that all knowledge about a concept is localised, ie stored in one spot. Thus, referential nets are highly redundant representations. This makes them costly in terms of storage capacity, because a description like chpos is stored thrice instead of only once, namely at the situation, the object, and the path refO, cf section 4.3. Yet, refNets are highly efficient with regard to access time, because once a refO is accessed all explicit knowledge about the entity is instantly available.‡ Since the main reason for using an incremental mode of processing for language production is to enable the human/system to speak fluently despite limited resources, incremental processing is mainly a means to save processing time, not storage capacity.

The second aspect is that refNets allow to change the representation, ie refOs, attributes, and designations can be inserted, deleted, or changed. For example, a name can be attached only temporally to the refO, eg the name ‘CK-314’ standing for the flight number can be replaced by another name for the return flight. Or, if the members of The Oscar Peterson Trio, represented by r8, change so that Niels-Henning Ørsted Pedersen (r9) and Terry Clark (r10) instead of Ray Brown and Herb Ellis now belong to the trio then r8 has the attribute parts([r5, r9, r10]).

* Not all knowledge about an entity is needed for a verbalisation, eg very often a name suffices to refer to an object (CK-314 in example (1)), and the other designations are not used. Consequently, an increment of an incremental preverbal message is determined by deciding that a refO will be verbalised. Then, in a separate step the knowledge about this refO that is needed for an adequate verbalisation is determined. This is part of perspectivisation, cf sections 6.0.2 and 5.4. The algorithm of how the selection of descriptions takes place is described in chapter 13.

† One may ask whether a representation really is contradictory if a process operating on it can resolve the conflict. Thus, from a logical standpoint it is more accurate to speak of conflicting representations. Examples are of the kind where a representation contains the propositions red(A) and green(A) and the knowledge that A can only have one colour. For cognitive systems this is quite plausible, eg one proposition may be due to perceived information while the other is inferred from previous knowledge. A system with such representations must be capable of dealing with the conflict somehow. Nevertheless, I will continue to speak of contradictory representations.

‡ In addition to this knowledge may be retrieved by following links to other refOs, eg knowledge about the owner of the plane in the above example. However, this only adds to the cognitive adequacy of referential nets.
4.4 Representations of the external world

Conceptual representations are internal representations. This means, they are particular to an individual (system or human). Thus, conceptualisation always uses subjective representations of states of affairs, not objective ones. For assuming the existence of objective representations one would need to know the situation-independent factors that would allow to build up such representations. Since humans are shaped by their experience, conceptualisation means to connect newly perceived with available information and to interpret it. So, an adequate conceptual representation must be an internal, subjective one. In particular, I do not consider discrepancies between the objective, ‘correct’ state of affairs ‘in the world’ (whatever that may be) and a subjective, ‘perhaps correct, perhaps not quite correct’ representation of the state of affairs within the system/human in this investigation.

Apart from the experience of the cognitive system, conceptualisation depends on a multitude of factors: available knowledge, context, attitudes, current state of the system/human, available resources, etc. These factors influence how conceptual representations are constructed and used. Consequently, conceptualisation can only be generalised to a certain extent. Hence, a rule like the cut-hypothesis mentioned in section 3.3 cannot be a way of finding events ‘in the world’. It is a way to explain why humans experience a succession of states as a coherent whole.

Yet learning issues are not considered here; so, isn’t there a contradiction? Well, cognitively adequate models should account for both, the typical performance as well as individual differences. A way to do this is to parameterise the model so that it shows a typical performance when the parameters are set to default values but by using other values it accounts for individual differences. The parameters of INC, which mainly model the available resources, not only have effects on the way representations are processed but also on the representations themselves.

Furthermore, there is the general problem to decide upon a correct representation in the first place. Representing always means abstracting away from certain details: it is (almost) impossible to represent the world in full detail; think of the absurd notion of a map in scale 1:1. And it would make (almost) no sense, because a representation always serves a purpose; for a language production system it is to produce language, in the case of an organism one major purpose is survival. So, correct can only mean correct with respect to a purpose. Glenberg (1997) highlights this point by stating that memory always is for something, especially representations of the current state of affairs. Since memory evolved to supply an organism with an internal representation of the perceived world, memory must be seen as embodied. According to Glenberg this means, firstly, that memory contains representations of the organism’s experiences, and, secondly, that it contains a collection of possible next actions in the environment. So, the memory of an individual contains representations that are, although dependent on the environment, representations for the individual, because they contain possible actions for the particular individual.

However, it is possible to identify downright erroneous representations. For example, it can be accepted if the plane is represented as an unidentified object,
but not if it is represented as an airport tower, eg due to a erroneous categorisation, because this will most likely lead to mistakes when the representation is used, eg when the represented scene is described verbally. (However, systematically occurring errors are an important source of insight into cognition.) In the online description of events a good criterion to evaluate the correctness of a conceptual representation is to compare the descriptions produced by the system, ie by the implementation of nC, with descriptions recorded in the verbalisation studies. If there are significant deviations it is justified to say that the representation is erroneous. Then again, it may be the processes working on the representation that are faulty, not the representation. Yet, such intricate issues are far beyond the scope of this investigation, even if it must be noted that internal representations and cognitive processing are strongly interdependent.

4.5 Event representations

Events are much harder to investigate experimentally than objects, and so concepts are usually taken to mean object concepts, not event concepts. Just like object concepts, event concepts serve as building blocks of conceptual representations. They fulfil the same function, are learned and used similar to object concepts. The main difference is that events are immaterial, ie they are more abstract.

In section 4.2 three kinds of refOs used in the conceptual representations were introduced: apart from situations (events) and objects there are spatial entities. The rationale for assuming spatial entities is analogous to the one for event concepts. Firstly, spatial entities like locations or directions are neither objects nor events. Nevertheless, they can be identified, eg a location close to a concrete entity or the direction north. Secondly, they function in the same fashion as the other kinds. Landau & Jackendoff (1993) present an argument that where something is in space is differently represented than what is located there.

Different kinds of refOs can be distinguished by their sort. In section 4.1 I already mentioned that sorts subdivide a conceptual representation into different layers and that sorts are arranged hierarchically; they form a subsumption hierarchy. The sort hierarchy for the motion event domain is depicted in figure 4.1.

The temporal relations between the situations in the conceptual representation are listed in table 4.3. The relations as I use them in the following are based on the ones proposed by Allen (1983, 1991) but see also van Benthem (1990). Since situations cannot only represent time intervals but also time points, the temporal relations are also given for relations between time intervals and time points and for relations between two time points. This approach follows the proposal of Vilain (1982). In the former case the relations are preceded by a p, in the latter case by pp so that, for example, the after relation becomes ·after for the relation between a time point and a time interval, after· for the one between a time interval and a time point,
4.5 EVENT REPRESENTATIONS

Figure 4.1: The part of the sort hierarchy (lattice) used in the motion event domain

<table>
<thead>
<tr>
<th>RELATION</th>
<th>INVERSE</th>
<th>EXAMPLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>X before Y</td>
<td>Y after X</td>
<td>XXX YYYY</td>
</tr>
<tr>
<td>X equal Y</td>
<td>Y equal X</td>
<td>XXX YYYY</td>
</tr>
<tr>
<td>X meets Y</td>
<td>Y met_by X</td>
<td>XXXYYY</td>
</tr>
<tr>
<td>X overlaps Y</td>
<td>Y overlapped_by X</td>
<td>XXX YYYY</td>
</tr>
<tr>
<td>X during Y</td>
<td>Y contains X</td>
<td>XXX YYYY</td>
</tr>
<tr>
<td>X starts Y</td>
<td>Y started_by X</td>
<td>XXX YYYY</td>
</tr>
<tr>
<td>X finishes Y</td>
<td>Y finished_by X</td>
<td>XXX YYYY</td>
</tr>
</tbody>
</table>

Table 4.3: Temporal relations

‘after’ when two time points are related. For a detailed discussion of this problem see [Guhe, Habel, & Tschander, in print].

In the remainder of this section I will show how the conceptual representation for the docking example is built up incrementally and point out particulars of the representation. The next section contains some remarks on additional knowledge that is required for generating incremental preverbal messages out of such a representation. The final state of the conceptual representation is shown in figure 4.2.

Keeping the different kinds of temporal relations apart is important as can be seen from the following example. If a punctual event occurs between two time intervals, these time intervals could not be in the meets relation, because if the time point were between the two intervals, there would be a gap between them. However, if the time point is part of one interval it must also be part of the other interval, which means that the intervals are in the overlap relation. Using different sets of temporal relations means that such interferences do not occur, and the temporal relations involving punctual events allow to perform temporal reasoning in an Allen-like style [Vilain, 1982].
Figure 4.2: Referential net for the example of the docking plane
To take up the motivating setting from section 1.2 again, when you look out of the window while waiting to board you will first see a configuration of objects:

\[
\begin{align*}
\text{gate} & \rightarrow r_1 & \eta x \text{gate}(x) & \text{[B21]} \\
\text{parts}([r_2]) & & & \\
\text{walkway} & \rightarrow r_2 & \eta x \text{walkway}(x) & \text{part_of}([r_1])
\end{align*}
\]

Thus, the initial state of the conceptual representation contains the two refOs that stand for the \textit{gate} and the \textit{walkway}. The fact that the \textit{walkway} is part of the \textit{gate} is represented by the attributes \texttt{parts} and \texttt{part_of} that establish the part-of hierarchy between refOs. (The \texttt{parts} attribute can be read as \textit{has parts}.) Both are list attributes, i.e., they have a list as argument. The designations represent that \(r_1\) can be described as a \textit{gate}, as \(B_{21}\), or as \textit{gate} \(B_{21}\) if two designations are used in the verbalisation and \(r_2\) as a \textit{walkway}.

When the \textit{plane} enters the scene in phase 1 as it is shown in figure 1.1 on page 4 three refOs are simultaneously added to the conceptual representation:

\[
\begin{align*}
\text{plane} & \rightarrow r_3 & \eta x \text{plane}(x) \\
\text{situation} & \rightarrow r_{41} & \eta x \text{chpos}(r_{41}, x, r_{6p}) \\
\text{path} & \rightarrow r_{6p} & \eta x \text{straight}(x)
\end{align*}
\]

The situation refO \((r_{41})\) and the three \texttt{chpos} descriptions that come with it connect the triple. The descriptions represent the fact that in the situation \((r_{41})\) an ob-

\begin{footnotesize}
\begin{itemize}
  \item Since the refNet is rather large I use a numbering of refOs deviating from the standard. It groups refOs belonging to one movement and refOs that have the same sort. \(r_{6p}\) is the path of the plane movement, \(r_{7w}\) the path of the walkway movement, \(r_{7f}\) is the location of the start of the walkway movement, \(r_{7f}\) the location of the final point of all movements. The subscripts of the situation refOs indicate the order in which the segments of the respective overall situation occur. Note that this is only a naming convention; using different names would not change the referential net.
  \item Actually, the conceptual representation should contain an additional refO that stands for the situation. I left it out, because it has no bearings on what follows. Note that it would be necessary if \(nC\) were to generate an utterance at this point like \textit{There is a gate}, because \(nC\) requires a situation refO to be able to start producing an incremental preverbal message.
  \item Using one of the two attributes would suffice. For example, if \texttt{part_of} were left out the information stored with \texttt{parts} could be used to compute the \texttt{part_of} information. Then, the whole refNet would have to be searched for \texttt{parts} attributes referring to the refO that would otherwise contain the \texttt{part_of} attribute, say \(r_2\). The refO \((r_2)\) is then a \texttt{part_of} all the refOs having such a \texttt{parts} attribute (only \(r_1\) in this case). Thus, one of the attributes is redundant and could be left out. Yet, incremental processing is about efficiency with respect to run-time, not storage use.
\end{itemize}
\end{footnotesize}
ject (r₃) moves along a path (r₆ₚ). Hence, the sort frame of chpos is (situation, object, location), cf also page 132. Since r₄₁ represents the first situation of the scene, it has no temporal relation. The –complete attribute stands for the fact that the situation is extended, i.e., takes place in a time interval, and that it is not yet completed. The situations that have the –complete attribute are the ongoing situations in a scene. The set of ongoing situations is important, because this is the set of situations for which the temporal relations are established that are introduced into the conceptual representation. Temporal relations between other situations must be computed if required.* The description ηₓ straight(x) of r₆ₚ represents the shape of the path of the movement. Note that paths are no real-world tracks but abstract, invisible spatial entities [Eschenbach, Habel, & Kulik 1999, Eschenbach, Tschander, Habel, & Kulik 2000]. The description ηₓ to(x, r₁) (sort frame (path, object)) says that the path is leading towards the gate (r₁).

This representation already allows to draw some inferences. First of all the description ηₓ to(x, r₁) says that the path is leading towards the gate. Together with the fact that in the situation a change of position is occurring (because of the chpos descriptions) this leads to the inference that there is an overall situation (r₄) in which the goal of the movement is the gate. The sort frame of goal is (situation, object).

\[
\begin{align*}
\text{situation} & \quad r₄ \quad \etaₓ \text{chpos}(x, r₃, r₆ₚ) \\
\text{parts}([r₄₁, r₄₂, r₄₃]) & \quad \etaₓ \text{goal}(r₁, x)
\end{align*}
\]

The next inference is already indicated in r₄ by the parts attribute: the complex situation brings with it additional simple ones. Although the plane reaches the gate only in phase 2 of the scene, this can already be anticipated (expected) in phase 1 by adding two refs to the representation.

\[
\begin{align*}
\text{situation} & \quad r₄₂ \quad \etaₓ \text{stop}(x, r₃, r₇ᶠ) \\
\text{punctual status(expected)} & \quad \cdot \text{finishes}(r₄₁) \\
& \quad \cdot \text{starts}(r₄₃) \\
\text{part_of}([r₄]) & \quad \etaₓ \text{be_at}(x, r₃, r₇ᶠ) \\
\text{situation} & \quad r₄₃ \quad \etaₓ \text{be_at}(x, r₃, r₇ᶠ) \\
\text{complete status(expected)} & \quad \cdot \text{met_by}(r₄₁) \\
& \quad \cdot \text{started_by}(r₄₂) \\
\text{part_of}([r₄]) & \quad \etaₓ \text{be_at}(x, r₃, r₇ᶠ)
\end{align*}
\]

* The set of ongoing situations is important in another respect: the temporal relation between two situations cannot be uniquely determined. This is particularly relevant in the case of concurrent events, which is elaborated further in Guhe, Habel, & Tschander [in print].
The fact that these two refO's were not actually perceived but are only anticipated is represented by the status attribute, which can have the values expected, regular, or discarded. Perceived or inferred refO's of which all parts are regular have status(regular). Expectations that were not fulfilled are discarded. For reasons of brevity I will only specify the status attribute in the following when it is relevant.

The refO's in figure 4.2 all have status(regular). More on expectations and the way they are computed and used in nC follows in chapter 10. At this stage also the first temporal relations are inserted (r41 is updated accordingly). r42 finishes the movement (the plane stops) and starts the the situation where the plane stands still (r43). Since r42 represents a punctual situation, ie it has no temporal extension, r41 and r43 are in a meets relation.

Adding these refO's into the conceptual representation causes the insertion of yet another refO (r7F), representing the location where the stop event takes place and where subsequently the plane is standing. This location is next to the gate and the walkway. Furthermore, it is the final point of the path r6p. Again, r1, r2, and r6p are updated with the according descriptions.

In phase 2 of the scene the plane stops next to the gate at the location already introduced as r7F. This has the effect that the two situation refO's r42 and r43, which were introduced with status(expected) now get status(regular). So, during this phase no new refO's are inserted into the conceptual representation, but the representation is modified. Note that I simply assumed without further comment that the system (nC) introduces these expectations. What is more, I assumed that the overall situation, represented by r8, is not expected. The reason for this is that I chose a 'medium willingness' of the speaker to generate expectations that was high enough for the first one but too low for the second. This willingness varies across verbalisations, cf part C. In nC it is modelled by the parameter doar (degree of agreement threshold), cf section 8.4. A low value of this parameter causes that no expectations are generated, with a high value nC would also create r8.

For the movement of the walkway in phase 3 and 4 the conceptual representation changes analogously. During these phases the docking refO (r8) is created as well. Note that a full representation would include a refO r5o for the situation when the walkway is at location r7S.

These refO's are not deleted from the conceptual representation. From a cognitive viewpoint this would mean that the model contained a 'forgetting mechanism', which nC does not. Doing so would allow utterances like At first I thought the plane would crash into the gate, but then... Without this, however, it must be ascertained that no discarded refO’s are verbalised and that no descriptions referring to discarded refO’s are used.
Let me conclude the discussion of event representations with two short remarks. Firstly, there are many other event sequences that can constitute the docking of a plane. Hence, I do not claim to have presented a full account of all possible conceptual representations for docking events. Secondly, the way I represent entities takes into account that they are the conceptual representations from which language is produced, i.e., they are representations for speaking. I will turn to this aspect now.

4.6 Representations for speaking

The conceptual representations I consider are constructed and used for the production of language. Taking up Slobin’s (1996) formulation: thinking for speaking requires representations for speaking. However, for producing incremental preverbal messages a speaker additional representations apart from the representation of the states of affairs (Levelt 1989), most notably:

- the common ground (shared knowledge of interlocutors, independent of the current discourse)
- the speaker’s own contributions
- the interlocutors’ contributions
- the information the speaker still wants to convey

In the setting used here not all of these representations are needed, or they can be left implicit. Since only one person/system is speaking no provision needs to be made for the contributions of the interlocutor. The common ground can be left implicit, i.e., no computations have to be performed with respect to whether the speaker and a particular interlocutor share some piece of information – a default interlocutor can be assumed instead. Generally, however, the common ground influences the generated utterances, because a speaker adapts his contributions in that he leaves quite a lot information implicit.

The information the speaker still wants to convey is information that must be represented in every setting; otherwise no utterance planning is possible. In order to produce a coherent discourse, not only isolated utterances, a memory for the speaker’s own contributions is needed. In this setting, this is the only information contained in the discourse memory, because there are no other interlocutors, whose contributions would also be part of the discourse memory. Thus, in the setting used here the discourse model consists of two items out of the four: the information the speaker still wants to convey and the speaker’s own contributions (the discourse memory).

The discourse memory is represented as a path through the conceptual representation that is represented as a referential net. I simply call it traverse, because the path is built up while traversing the referential net. It consists of the refOs that have
been used in a preverbal message. For instance, in the generation of an incremental preverbal message for an utterance like (10a) the refOs $r_4, r_3, r_6P, r_1$ are taken – in this sequence – and added to the traverse.

(10) a. *Ein Flugzeug bewegt sich auf ein Gate zu.*

‘A plane is moving towards a gate.’

b. 

<table>
<thead>
<tr>
<th>situation</th>
<th>$r_4$</th>
<th>$\eta x\ chpos(x, r_3, r_6P)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-complete</td>
<td></td>
<td>$\eta x\ goal(r_1, x)$</td>
</tr>
<tr>
<td>plane</td>
<td>$r_3$</td>
<td>$\eta x\ plane(x)$</td>
</tr>
<tr>
<td>path</td>
<td>$r_6P$</td>
<td>$\eta x\ to(x, r_1)$</td>
</tr>
<tr>
<td>gate</td>
<td>$r_1$</td>
<td>$\lambda x\ gate(x)$</td>
</tr>
</tbody>
</table>

An example of the information that is stored in the traverse after this is that the gate was described as *the gate*, not as *B21* or *gate B21*. Note that as part of the generation of the preverbal message the $\eta$-operator in the conceptual representation is changed to a $\iota$ in the preverbal message, which results in *the gate* instead of *a gate*.† At the end of chapter 3 I already mentioned that the mechanism with which an incremental preverbal message is generated consists basically of attempting to ground chosen designations. Here, the designations $\eta x\ chpos(x, r_3, r_6P)$ and $\eta x\ goal(r_1, x)$ of $r_4$ refer to other refOs ($r_3, r_6P, \text{and } r_1$), for which designations must be chosen, because the refOs are already ‘announced’ by sending $r_4$ to the formulator as first increment of the incremental preverbal message. This is a consequence of the early commitment strategy of iNC, cf also the definition of Extended Wundt’s Principle (term 5.2 on page 82).

The information to be conveyed is stored in the traverse buffer. The traverse buffer contains pointers to situation refOs that were selected by the selection process but for which no preverbal messages have been generated yet. Until refOs in the traverse buffer are verbalised they are subject to change, ie the decision to verbalise a refO can be retracted by the selection process. Therefore, they are stored in a buffer. However, since the function of the traverse buffer depends strongly on the notion of incrementality used in this work, I delay its detailed description until part C. Let us now first turn to the issue of incrementality.

* This is not completely correct. In fact, each time a refO is used in a preverbal message a new refO is created that stores the current state of the refO at that point of time and the information used in this verbalisation. Otherwise the discourse memory could contain each refO only once and only in its current state, ie not in the state in which it was verbalised. I will give the full account in chapter 3.

† For the next incremental preverbal message it can be tested whether it is adequate to use a pronoun for referring to the gate, which in this case it would be not, because *it* would refer to the plane ($r_3$).
INCREMENTALITY
Incrementality is the second major topic of my investigation. In contrast to language production, incrementality is mainly an issue of informatics and AI. It originated in an area of compiler theory, where it was introduced in the 1960’s to designate a processing mode in which it is possible to apply modifications to a compiled program without compiling the whole program again. Instead, only the modified parts are recompiled and integrated into the existing program. The goal was to create more interactive and efficient compilers. Well-known examples of incremental algorithms are spell-checker algorithms of word processors: only modified words are checked by the algorithm, not the whole text.

The notions of incremental processing put forward in the literature focus on the externally observable behaviour of computing devices, usually systems. The characteristic behaviour is that incremental systems produce output before all input that may have effects on the correct and complete computation of the corresponding output is available. My focus differs in two respects. Firstly, it emphasises the fact that each time a new input increment is read in it is interpreted with respect to the current state of the system. Secondly, a corresponding output increment is computed with respect to the input increment and the currently available knowledge.

The aim of this chapter is to discuss incrementality with respect to standard issues of informatics. In chapter 6 I will provide a formalisation of these issues. This formalisation provides a starting point for developing incremental models. The structure of this chapter is as follows. In the first section I will introduce the most important properties of incremental processing. In section 5.2 I will report on distinctions between different kinds of incrementality put forward in the literature. Section 5.3 introduces the terms I use to describe the structure and functioning of incremental models, before in section 5.4 I discuss the dimensions along which such incrementally operating models can vary. The variety of incrementality used in INC will be described only in section 7.1 because for this, the issues of resource limitations and of unified cognitive architectures are needed, which are the topic of chapter 7. The topic of section 5.5, finally, are the requirements that a representational formalism must fulfil so that it can be used for incremental processing. In particular I will present semantic underspecification formalisms as a promising alternative to referential nets for representing incremental preverbal messages.
5.1 Approaching the phenomenon

Incremental processing is known in informatics for quite a long time considering the youth of the discipline, namely since the 1960's. According to Finkler the first pioneering publications are by Lock and Ryan, Crandall, & Medwedeff. It originated in the area of compiler construction as a means to change the compiled program only according to the changes in the source code instead of recompiling the whole program. Thus, the main advantage of incremental processing is the reduction of complexity of computations because results of previous computations are reused. Put more generally, incremental processing means that only the changes are computed with respect to the current state of the knowledge that is result of the last run of the incremental algorithm. Note that this makes it especially suited for models like nC that are situated in a dynamic environment.

My notion of incrementality additionally reduces the complexity of computations in that not all available knowledge is considered when a new piece of information is acquired. Instead, that part of the available knowledge used for the previous computation is the starting point for evaluating the new information. The advantage is that in most cases this enables the system to correctly include the new information into its knowledge. Accordingly, only in a minority of cases a global search of the all knowledge is necessary.

The notion of incremental processing prevalent in language production is more specific than the one in informatics. It was first put forward by Kempen & Huijbers, see also Kempen & Hoenkamp, De Smedt & Kempen, and Levelt. It has two major properties: it is a piecemeal processing mode and it uses a cascade of subsequent processes, cf chapter. Piecemeal, sometimes also called chunking, means, firstly, that computations take place before all information that may be relevant for the complete and correct computation of an output is available. Thus, not all input is available when the generation of output commences. Piecemeal processing means that it does not consist of a continuous flow of information but of discrete increments (fragments/pieces/chunks).

The incremental mode of processing can be achieved by a number of different architectures. The most prominent of these is the so-called cascaded architecture, a symbolic representation of which can be found in figure. The cascade metaphor represents the idea that, just like a water stream in a water cascade, a stream of increments exists on different computational stages at the same time; and on each stage a different increment of the stream is processed. After a process finished its computations for an increment, it ‘splashes’ down to the next process. This over-
5.1 Approaching the Phenomenon

Figure 5.1: A piece of information running through a cascade of processes

all design is captured by the phrase *parallel processing of a sequential information stream*. In figure 5.1, the information stream is represented by the arrows. It is processed by processes $p_1$, $p_2$, and $p_3$. The dots with the annotated $i_1$, $i_2$, and $i_3$ can be read in two ways. Under the first perspective, the figure shows a snapshot of the system and they stand for three different increments belonging to the same information stream at a particular point of time. Under the second perspective, the x-axis represents the flow of time and the dots stand for the same increment at three subsequent points of time. Apart from being resource-effective, such cascaded systems have the advantage that they are able to keep up with the pace that input arrives and to produce output at a constant rate. Moreover, they are capable of simultaneously reading input and producing output. They are therefore suited for the data-driven online scenario I am investigating.

There are also substantially different notions of incrementality proposed in the literature. To give just one interesting example, Liu (2000) presents a program analysis and transformation technique she calls *incrementalisation* with which she derives an ‘incremental’ function (algorithm) from a non-incremental one. For the algorithms generated this way, she shows that they are more efficient than their non-incremental counterparts. Note that the main incentive in this approach is to save resources (run-time) as well. However, this technique is really one for storing (intermediate) results so that they need not be computed again when the function is called with the same arguments.

In contrast to this, incremental processing as I present it also improves runtime if a result has not been computed before. This mainly hinges on the fact that all processing is the processing of an increment with respect to the current state of the available knowledge. This increment is the *focussed element*. Since, furthermore, the knowledge is not unstructured, the relevant knowledge for interpreting the focussed element is surrounding it. For referential nets this means that the relevant knowledge is either directly connected to the focussed element or not many links apart. This relevant knowledge is the *local context* with respect to which the focussed element is interpreted.

This method, however, may also *increase* run-time if no or only a bad result is
found for the local context, because in this case a new local context is established and the incremental algorithm is executed again. This may result in a complete search of the knowledge. Thus, it is not even ascertained that performance is as good as it would have been if the whole available knowledge were taken into account right from the start. Yet, this effect is a strong argument in favour of the cognitive adequacy of the processing mechanism, because it allows to model attention: the focussed element is the element attended to. What is more, the increase in run-time shows an analogy to the phenomenon of priming. The elements in the local context for a focussed element can be regarded as pre-activated, which facilitates the response to new input, because less information has to be taken into account. Hence, the run-time for the computation is reduced. If the computation fails, however, the response time increases, because then different local contexts have to be tried, and in the end all available knowledge is used. However, these extreme cases cannot be detected before processing commences.

I want to conclude this section by introducing the principle on which my notion of incremental processing is grounded. The principle is introduced by [Levelt, 1989: 26]. He names it Wundt’s Principle after Wilhelm Wundt, who first proposed such a mode of processing. It basically states that processes in an incremental system are hungry or eager, ie start processing as soon as possible.

**Term 5.1** Wundt’s Principle. Each processing component will be triggered into activity by a minimal amount of its characteristic input.

There are two important points about this principle. Firstly, the characteristic input (cf page 24) must be defined individually for each processing component. Secondly, it must be identified what constitutes a minimal amount for a processing component. It can be said, however, that a minimal amount is an increment, which in turn raises the question of the size of increments. Note that saying that the minimal amounts of input are increments does not mean that all increments constitute a minimal amount; increments can be of different sizes.

In the following I will use a stronger, extended version of Wundt’s Principle. The extension consists in that components not only start processing as soon as possible but also produce output as quickly as possible.

**Term 5.2** Extended Wundt’s Principle. Each processing component will be triggered into activity by a minimal amount of its characteristic input and produces characteristic output as soon as a minimal amount of output is available.

Using Extended Wundt’s Principle means that the processes do not evaluate their output, ie they make no decision whether the overall output would benefit if it was not handed over to the subsequent process immediately but if the process waited longer until, say, the quality of the output can be judged more reliably. There is various support for this view, eg by [Bock, Irwin, Davidson, & Levelt, 2003] and...
by De Smedt (1990a) citing work by Hoenkamp states that ‘what can be uttered must be uttered immediately’. It is also nourished by results of studies on language processing, eg Altman & Kamide (1999) and Crocker & Brants (2000) – although it is not clear whether language production and language comprehension work according to the same principles. Extended Wundt’s Principle stands in opposition to the principle proposed by Kilger & Finkler (1995) which says to ‘output as soon as necessary, not as soon as possible for subsequent output increments’ (ie after the initial output increment) so as to optimise quality and reliability of the output.

5.2 Kinds of incrementality

In this section I will describe distinctions of different kinds of incrementality that are put forward in the literature, before in the next section I will present my account of incrementality. Note that in this section I will adopt formulations to my use of terminology if necessary. Since I am concerned with language production I will mainly discuss the literature from this field. The following distinctions serve to characterise possible variants of incrementality and to distinguish the kind of incrementality used by INC from other proposals – which may be adequate for other purposes.

**Left-to-right (LR) vs full.** Wirén (1992) distinguishes full incrementality from left-to-right (LR) incrementality. In full incrementality an input increment can affect changes at any place in the knowledge representation, while in left-to-right incrementality this can only happen to the ‘right’ of the knowledge representation. Since Wirén is interested in incrementality from an **NLP** perspective, he considers LR incrementality to be the important variant in parsing. Full incrementality has mainly applications in information systems, eg in text editors.

Full incrementality is the original kind of incrementality that was used in informatics (Wirén 1992, Finkler 1997). As already laid out in section 5.1 the impetus for developing methods of incremental processing was to save resources, in particular run-time. An incremental compiler must allow changes anywhere in the program. The problem caused by this method is to discern the parts of the program that needed to be altered from those that remain unchanged. In A1 – replacing **program** by **knowledge representation** – this is tantamount to the **frame problem**, cf the remarks in section 7.4. LR incrementality can be regarded as special case of full incrementality that is particularly useful for the processing of language. The restriction is that the changes can occur only on a particular point in the knowledge representation: to the ‘right’ of what was processed before.

Neither of the two forms of incrementality is desirable in its pure form for con-
ceptualisation: InR incrementality is severely limited, because it allows changes only at a defined point in the knowledge representation, which cannot be ascertained for conceptualisation. For example, in the taxiing scenario a second plane may appear at any time, which cannot be interpreted with respect to the most recent movements of the first plane, ie it cannot be interpreted by attaching the new information to the ‘right’ of the current knowledge representation. Full incrementality, on the other hand, is very costly in terms of resources, because changes can occur anywhere, affecting any element of the representation. Therefore, in mC a combination of both is employed: although the conceptual representation can be changed anywhere, the access to it is not random access. Instead, it starts at the focused element and considers a local context first. If this fails other local contexts are constructed. In case of repeated failures this method is more costly than full incrementality, cf section 5.4.

Massive vs moderate. Hildebrandt, Eikmeyer, Rickheit, & Weiß (1999) make a distinction between massive and moderate incrementality. In massive incrementality each increment is processed as quickly as possible. Hence, it is compliant with Extended Wundt’s Principle. Moderate incrementality means that output is not always immediately generated. Instead, the process may wait some time, store the intermediate result internally, and wait for the next input increment. Thus, buffering is necessary for moderate incrementality. If the system design ascertains that there is no danger that increments get lost (or that this does not matter) then massive incrementality can work without buffering.

An example of massive incrementality in conceptualisation is a selection strategy that immediately decides whether to verbalise a newly perceived event. For example, as soon as a plane movement is perceived the conceptualiser can decide to describe it. The decision can be reverted afterwards – at least until the corresponding verbalisation is generated. Nevertheless, it is made without delay. In contrast, a conceptualiser using moderate incrementality evaluates whether the decision should already be made or whether waiting will pay off, eg until it becomes clear where the plane is heading.

This distinction stems from the area of language comprehension, where it poses eminent problems: some expressions cannot be parsed unequivocally when information is missing – information that is still to follow. However, every incremental model is subject to the general problem. If an ambiguity arises there are principally three ways to handle it:

1. delay the decision until all information is available,
2. choose one possibility,
3. generate all possibilities in parallel.

The first solution seems to be the most attractive one: the system does not generate any errors, and no corrections or recoveries are required. This is the route taken in most NLG and NLP systems. However, it is the opposite of incremental processing,
because it contradicts the idea of a pace-keeping way of processing. If the system waits until all information is available no output can be produced before this is the case. For systems (and humans) situated in an environment to which they must react this is no viable solution, because input is infinite, and there is no way to be certain that following input does not invalidate output generated earlier on. Furthermore, assuming that all information can be available is a strong idealisation in the first place. Such approaches do neither take into account the limitations of time and knowledge that humans must cope with nor the fact that they use cues from the available information to make decisions instead of global optimisations, cf the discussion of bounded rationality in section 7.2. Thus, this is no solution for incremental processing at all.

The second solution seems to be less attractive at first sight, because reverting a decision may be costly in terms of resources. It is the best solution, though, if

1. the goal is to model characteristic human errors, eg talking oneself ‘into the corner’;
2. reverting a decision is comparably cheap, or
3. the decision procedure is so accurate that only seldom decisions have to be reverted, so that making the decision early on reduces the overall resources and/or enhances the overall processing speed.

The third solution may seem equally unattractive, because often there are not only two or three possibilities but many more. Sometimes there are even infinitely many possibilities, and to pursue them all is impossible. However, there are cases in which the number of possibilities is small so that it is reasonable – advantageous in terms of resources – to generate them all, cf the discussion on the number of increments that are processed in parallel on page [104].

Empirical evidence that massive incrementality (Extended Wundt’s Principle) is an adequate processing principle for cognitive models is provided by Altman & Kamide (1999), Crocker & Brants (2000) and by Hildebrandt et al themselves. Consequently, INC uses mostly massive incrementality.

*Word-based vs constituent-based.* Apart from the massive/moderate distinction Hildebrandt, Eikmeyer, Rickheit, & Weiß (1999) also distinguish word-based from constituent-based incrementality. In incremental processing it corresponds to the ubiquitous problem of identifying the increments that are processed. In word-based incrementality the increments are mainly words, in constituent-based incrementality they are mainly constituents of sentences. I talk of mainly, because incremental processes should be somewhat flexible with regard to the question of what increments an can handle.

Although words and constituents play no role on the level of the conceptualiser two things can be learned from this distinction. Firstly, Hildebrandt et al find that a massive, constituent-based incrementality is supported by results from linguistic as well as psycholinguistic investigations, ie it is linguistically sound and cognitively adequate due to empirical studies. Although unfortunately they only appeal...
to the linguistic notion of constituents in general and provide no detailed criteria of how constitutes are defined, constituents are (mostly) larger increments than words. This means, the characteristic input and characteristic output of incremental processes need not necessarily consist of the smallest increments possible. The selection process deciding on sub-intentions is a similar case in the conceptualiser, cf section 3.2. These increments are rather large. Since sub-intentions are about whether a situation is described or not, one could call this situation-based incrementality. Secondly, the output generated by incC is rather close to constituent-based incrementality, because a refO that is an increment of an incremental preverbal message corresponds roughly to a constituent in its informational content. This could be called refO-based incrementality.

Partial incrementality. Finkler [1997] proposes a definition of the behaviour exhibited by incremental systems. In contrast to the notion of incrementality presented so far, which is based on a set of processes arranged in a cascade, Finkler introduces a notion of incrementality based on a single system. He distinguishes three kinds of processing modes ("Verarbeitungsmodus"): incremental ("inkrementell"), partial incremental ("partiell inkrementell"), and sequential ("sequentiell"), cf figure 5.2. The defining property of the incremental processing mode is that the generation of output starts before the input is complete; so it corresponds to what I call incremental behaviour, cf term 5.16 on page 99. Partial incrementality must have temporal overlap of input and processing, or processing and output, or both, but no temporal overlap.
overlap of input and output, because that would be incremental, not partial incremen- 
tal. The sequential processing mode has no temporal overlap of input, process-
ing, and output, thus, all relevant input for computing the corresponding output 
is completely available when processing starts, and the processing is finished before 
any output is generated.

Although this distinction can certainly be made I cannot see its usefulness – at 
least not for the investigation at hand –, because the cognitively interesting systems 
are those that are capable of ‘proper’ incrementality. Partial incrementality is only 
a special case for such systems. Additionally, this distinction is rather weak in its 
claims about cognition, because it is only concerned with the externally observable 
behaviour but not with the internal mechanisms that bring it about. In other words, 
the claims are very general, because they are only concerned with the mapping of 
input to output and with the degree systems and humans show the same externally 
observable behaviour. See also [Lewis 1999] on this problem and the discussion of 
different levels of architecture by [Newell 1990].

Qualitative vs quantitative. [Finkler 1997] also distinguishes qualitative and quan-
titative incrementality. A system works in a qualitative incremental fashion if it al-
ways obtains a complete input, ie if the input data has changed, the system not 
only receives the new information but all information, including all old informa-
tion. The assumption is that the quality of the input information increases over the 
course of time. A quantitative incremental system hands on only new or changed 
information. This means there are two different kinds of increments, those that con-
tain new information and those that are updates (corrections /enhancements) for 
previous increments. In qualitative incrementality there is only one kind of incre-
ment: a complete representation of differing qualities.

For nC qualitative incrementality would mean, for example, that the construc-
tion process always receives a complete representation of the scene from the PPU. 
Thus, it would receive not only the information about the current position of the 
plane – as in a quantitative incremental system – but also about the fact that there is 
a gate and a walkway, that the colour of the plane is black, and so on. As I already ar-
gued, I do not regard this as a cognitively adequate way to model conceptualisation. 
Therefore, nC uses quantitative incrementality.

What is more, while quantitative incrementality is just another name for piece-
meal processing, qualitative incrementality can hardly be brought into accordance 
with piecemeal processing. The reason is that the input/output does not consist 
of pieces (increments, chunks, new information) but all information, ie such sys-
tems are somewhat different. Indeed, they have more in common with anytime 
algorithms, which are, however, closely related to incrementality, cf section 7.3. The 
most notable difference is that anytime algorithms produce output of better quality 
the more time they have available, while incremental processing is used for pro-
cessing information at a constant rate, in particular to keep pace with the input. 
So, using the well-established notion of anytime computation instead of qualitative 
incrementality is much more accurate.
Bounded vs unbounded. Ramalingam & Reps (1993: 503) put forward a distinction of bounded and unbounded incremental algorithms. An incremental algorithm is bounded if its processing time depends only on the size of the input increments. This can mean two things. Firstly, if all results are stored in one representation, eg a conceptual representation, the time for changing this representation solely depends on the input size. Secondly, if the output is given to the next component directly, it is the time that is required to compute this output. If this dependency of processing time on the increment size is not given, an incremental algorithm is unbounded.

The incrementality of INC is unbounded, because apart from the size of the input the time it takes to process an input depends on the reasoning processes that are carried out. First of all this means that the computations required for finding a complex concept for the input (the categorisation task, cf section 4.1) depends on the content of the input, not its size. Note that bounded incrementality is a different issue than bounded rationality, cf section 7.2. INC definitely is a boundedly rational model.

Linear vs hierarchical. Bock, Irwin, Davidson, & Levelt (2003) differentiate linear and hierarchical incrementality. In linear incrementality an incremental process generates one increment after another without making any plans concerning future increments. In hierarchical incrementality the preparation of an increment may involve making such plans, ie following increments can partially (or completely) be prepared, but they are not yet generated. In other words, the hierarchical structure is generated, before it is filled with increments.

In INC mostly linear incrementality is used, eg the selection process only considers one increment at a time and does not generate structures for futures ones. Hierarchical incrementality is used at an important point, though: the initial increment (refO) of an incremental preverbal message induces a framework for the following increments in that it specifies what other refOs have to be verbalised. Moreover, each refO that is an increment of an incremental preverbal message may induce additional frames, ie announce further refOs.

Free-format vs preformatted. The last distinction I report here is similar to the previous one. It is put forward by Kempen: free-format vs preformatted incrementality. In free-format incrementality the order of increments is established while they are generated, ie increments are generated as they become available. In preformatted incrementality a sequence of output increments is generated by editing a previous sequence of output increments. The hierarchical structure and linear order of a previous sequence are simultaneously available. Some increments from the old structure are replaced by other ones; the remaining increments and their linear

* If this reasoning process is not carried out, which can be achieved by switching off the concept matcher, cf chapter 10. INC may indeed use bounded incrementality. Yet, this is only a conjecture.
† This idea was presented at a meeting of the DFG priority programme Language Production on 9 September 2002 in Frankfurt.
order are reused in the new increment sequence.

Yet, although the hierarchical structure and the linear order of an increment sequence are simultaneously available, the editing and generation of output takes place piecemeal, because an incremental system does not have the ability to process structures at once. In the case of the human language production system this means that even if a whole utterance is available it cannot be generated as a whole.

Kempen is mainly concerned with syntactic priming, where he finds empirical evidence for preformatted encoding. Since preformatted encoding also occurs in self-corrections (called self-repairs by Kempen) and in coordinations, both modes of operation are relevant for conceptualisation as well. (The similarity of self-corrections and coordination structures will exploited in section 5.5.) The most relevant point for inC is that, according to Kempen, a speaker prefers to reuse existing structures for pointing out the meaning contrasts between the current and an earlier utterance, ie he prefers preformatted incrementality. This has consequences for the processes selection and PVM-generation (preverbal message generation). Consider the following example, where the structure of the first utterance is reused in the second one and points out the meaning contrast:

(1) a.  
*Ein Flugzeug fährt auf Gate B21 zu.*  
'A plane moves towards gate B21.'

b.  
*Ein anderes Flugzeug fährt auf Gate B23 zu.*  
'Another plane moves towards gate B23.'

5.3 A blueprint of incrementality

Based on the characterisation of incrementality in section 5.1 and the kinds of incrementality proposed in the literature (section 5.2), I am going to define the terms that are the groundwork for what follows. They will be formalised in chapter 6 which in turn constitutes the skeleton of the description of inC in part C. As I already stated, the main intent of the terms is to explain the incremental mode of operation (incremental processing), not to describe externally observable behaviour of incremental models and systems. Furthermore, they are mainly for practical use, ie many terms might can be cast in a formally more elegant way without changing the substance of their meaning. However, since I consider this to be a kind of blueprint for incremental models I am more concerned with considerations on how the notions can be applied when building such an incremental model.

In a nutshell, the notion of incrementality that I will put forward in the following is this. A number of processes arranged in a fixed sequence (pipeline, cascade) operate on an information stream that is read in by the first process of the sequence and output by the last process, cf figure 5.3. Each process has exactly one predecessor and one successor – except the first and the last process. They all operate on a shared memory containing the available knowledge on which they all have different views. This encompasses incremental models in which each process has its
own representation. The pieces of information sent from one process to the next are increments.

At first it must be noted that incrementality cannot be understood as an abstract phenomenon; it is a property of something that carries out computations: models, systems, agents, processes, algorithms, functions, procedures, and so on. Thus, any device that is capable of computing and behaving in some way, ie is capable of producing output for input, can be incremental. The central notions in the following are incremental algorithm, incremental process, and cascade, which is the combination of incremental processes in a model, system, agent, or any means of encapsulating multiple incremental processes.

**Term 5.3 Incrementality.** Incrementality is the property of a model, system, algorithm, or process to compute information in a piecemeal manner.

Since incrementality can be adequately be described as piecemeal processing, the question now is what the pieces are. First of all, they are called increments.

**Term 5.4 Increment.** An increment is a piece of information that is the input of an incremental process (input increment), the output of one (output increment), or both.

(Incremental processes are defined below.) Formally, increments are defined as words of a formal language. For rnc this formal language is defined within referential nets (chapter 4). The words serve as input and/or output of incremental processes. According to their function they are called input and/or output increment. Thus, an increment is defined with respect to the processes of which it is the input and/or or output. This can easily be seen when considering that an increment for visual processing, for instance, must be different from an increment.
for phonological processes. An example in INC are the input increments to the selection process, which are the refOs that the construction process inserts into the current conceptual representation (CCR), cf chapter 11.

Since increments always represent only a piece of knowledge at a given point of time, it must be possible to change already generated output increments. Such changes are necessary when later computations, e.g. due to new input, show that a previously generated increment is false, faulty, or can be improved. For these cases update increments are needed.

Term 5.5 Update increment. An update increment is an increment that updates an increment previously read or sent by an incremental process.

Update increments are a special kind of input and output increments. So, speaking of increments includes update increments. There are two major cases for update increments. In the first a previous increment is modified; in the second an incremental process tells a subsequent one to ignore a previous increment. If the reading process has already made computations based on this increment, it must to perform a recovery. If the reading process has not used the increment up to then it can simply be dropped (forgotten).

Term 5.6 Increment stream. An increment stream is an ordered sequence of increments that changes over time. Increments can be appended to and popped from the increment stream.

(Popping means that the head of the increment stream is read and simultaneously removed from the increment stream.) Thus, an increment stream is a FIFO buffer of unlimited size containing increments, but see also the discussion of increment buffers (term 5.13). In particular, an increment stream serves to connect two incremental processes in order to form a cascade, see term 5.14. The four conceptualisation tasks identified in part A are realised in INC as incremental processes that are connected by increment streams.

Term 5.7 Incremental process. An incremental process is a process that behaves incremen tally. It reads input from an increment stream, writes its output to another increment stream, and recursively executes the following two steps: (1) it determines a local context and (2) calls an incremental algorithm with respect to the available knowledge. The recursion ends if (a) a new input increment is available or (b) the result(s) cannot be improved. In the second case the process suspends itself until a new input increment is available. An incremental process runs in an infinite loop until it is explicitly terminated from the outside.

* At least this must be possible in models using non-monotonic incrementality, cf section 5.4.
† Since there is usually no direct feedback in incremental models, the sending process cannot know whether the update was successful. If this is necessary increment buffers (term 5.13) can be used.
This notion of incremental processes emphasises the proximity to anytime processing. Other variants are conceivable, especially processes that do not execute an incremental algorithm in an infinite loop, e.g., processes that execute the algorithm repeatedly until a termination condition is met. Such processes are not terminated from the outside but terminate themselves. Another variant are incremental processes that execute their incremental algorithm only once for each input increment. If the algorithm is reliable a lot of processing time can be saved this way.

The property of explicit termination from the outside is due to the fact that the incremental algorithm is executed recursively (or in a loop) until a new input increment is available. If there is no input increment and the result cannot be improved further, an incremental process suspends itself. Hence, the process runs infinitely unless it is terminated by an external signal. The signal can, for example, consist in the information that no further input increment will follow. Especially this property makes my notion of incremental processing similar to anytime processing.

The interplay between the next four terms, model knowledge, available knowledge, local context, and focussed element that describe the processed knowledge, is depicted in figure 5.4. The implicit assumption in this figure is that it is a representation using referential nets; yet, other representations can equally be used with the following notions.

---

* Distinguishing different kinds of incremental processes is something different than the distinction between qualitative and quantitative incrementality, because this distinction identifies the kind of output of an incremental process, not conditions for its termination.

† Such a method is similar to the one reason decision making of Gigerenzer, Todd, & the ABC Research Group (1999).
5.3 A Blueprint of Incrementality

**Term 5.8 Model knowledge.** The model knowledge is the declarative knowledge of an incremental model.

In iNC the model knowledge consists of the conceptual representation in the ccr and of the concept formation rules in the concept storage (cs). The part of the model knowledge that is accessible by an incremental process and its incremental algorithm(s) is its available knowledge.

**Term 5.9 Available knowledge.** The available knowledge is the model knowledge that is accessible by an incremental process and its incremental algorithm(s).

An incremental process usually does not have access to all knowledge of the incremental model but only to a part. So, the available knowledge is not the complete knowledge of the incremental model. The view that an incremental process has on the knowledge of the model is managed by the model, ie the model determines what knowledge is seen by an incremental process.

**Term 5.10 Focussed element.** A focussed element is an element of the representation of the available knowledge. Each incremental process has one focussed element. It is the entry point to the available knowledge of an incremental algorithm executed by the incremental process. An incremental algorithm starts evaluating the available knowledge from this element.

The focussed element is the ‘hot spot’ of the available knowledge of an incremental process. It is either resulting from the last recursion of the incremental algorithm, or it is a new input increment of the incremental process. For example, after iNC’s construction process inserted an element into the ccr it hands on a pointer to this element to selection, which makes the element of the ccr pointed to the focussed element of the next call of selection’s incremental algorithm.

In referential nets focussed elements are usually refOs. However, this is not the only possibility. For example, a refO-description chosen for verbalisation by rvm-generation can repeatedly be the focussed element when the refOs referred to in this description are verbalised one after the other.

**Term 5.11 Local context.** The local context is that part (subset) of the available knowledge that is considered by an incremental algorithm in its computation. It is a connected sub-part of elements in the available knowledge around the focussed element. In each recursion of the algorithm the local context is different. It is determined by a heuristic that is specific to the incremental algorithm for whose call it is generated.

Each incremental process has only one focussed element. This is due to the fact that although an incremental process may have the choice between different incremental algorithms, it can only execute one algorithm simultaneously.
The basic idea of local contexts is that only the part of the available knowledge surrounding the process’s focussed element is used by an incremental algorithm. Thus, it is defined as interconnected sub-part of the available knowledge around the focussed element. Local contexts reduce the complexity of the algorithm’s computations, because less knowledge is considered. So, there are two reductions of the knowledge considered: the reduction of model knowledge to available knowledge and of available knowledge to local contexts. Local contexts are similar to the notion of locality by Ghidini & Giunchiglia (2001), who propose two principles:

Principle 1 (of Locality). Reasoning uses only part of what is potentially available (e.g., what is known, the available inference procedures). The part being used while reasoning is what we call context (of reasoning);
Principle 2 (of Compatibility). There is compatibility among the kinds of reasoning performed in different contexts.

The locality principle captures more or less what I call local context. The problems of compatibility are quite hard to solve, eg the problem that modifications of the common representation performed by a process must not lead to a situation in which it is impossible for another process to process this part. Nevertheless, I will not consider these problems, because my main interest is in the processing mechanisms. Hence, the builder of the incremental model has to ascertain compatibility.

Since, the local context is determined by a heuristic, it is not ascertained that globally optimal local contexts are determined in the sense that, firstly, the local context captures all knowledge that is relevant for the computations of the incremental algorithm and, secondly, that the next execution of the incremental algorithm leads to results consistent with the available knowledge. Here is an example of how an inconsistency can arise. Assume that inC gets the information from the ppu that a plane is standing at a gate. As long as this information is not changed by new information this is held to be true. Consider further that inC may not be told that the plane is leaving its position and moves out of the observed area. A reason for this may be that the ppu is occupied with other movements and there is too little time. If now another plane is moving to the position at the gate where the first plane was standing inC may not notice that according to its conceptual representation this position is already occupied by another plane. This can happen, because the local contexts are established starting from the moving object and without checking locations. Now there are two planes occupying the same location, which is impossible in the real world. Since such inconsistencies are uncommon among humans observing the reduced setting used here, I do not consider them further.

Inconsistencies are not acceptable for all incrementally working systems, first of all those that must have consistent representations, eg compilers or text editors.

\* The frame problem – determining which facts are needed for a computation and which facts are affected (changed) by the computation – that overshadows such reductions is solved by assuming that it does not need to be solved due to the goal of cognitive adequacy, cf section 7.4.
However, inconsistencies contribute to the cognitive adequacy of my approach to incremental processing, because humans cannot take into consideration all facts known to them due to resource limitations, cf also section 2.6 on bounded rationality. As a consequence, humans do not have consistent representations, i.e., representations without contradictions or representations that cannot lead to contradicting inferences or deductions. Thus, mechanisms must be provided for either resolving inconsistencies or for allowing to continue processing in spite of them.

The above definition leaves open the possibility that local context and available knowledge may be identical. This is the case in full incrementality as opposed to LR incrementality. The view corresponding to full incrementality is commonly assumed by approaches to formalising dynamic systems, e.g., [Reiter 2001].

The heuristic determining the local context can also be seen as part of the incremental algorithm, in which case the heuristic would be the algorithm’s first computation. Yet, this makes no substantial difference. I chose the other possibility, because in this way the view that an incremental algorithm has on the representation is not determined by the algorithm itself but by another instance. It therefore corresponds closer to the idea of views on representations.

The notions of focussed element and local context correspond to Levelt’s focal centre (focussed element) and focus (local context, [Levelt 1989: 119f]). However, the way I defined them is much more general, not specific to a task. Thus, I conjecture that these notions play an important role in all incrementally working processes, in particular those that aim at cognitive adequacy.

After having defined all terms required for the execution of an incremental algorithm I now turn to this notion.

**Term 5.12 Incremental algorithm.** An incremental algorithm is an algorithm that has the following properties: (a) it obtains a triple of focussed element, local context, and available knowledge as input; (b) it only considers the local context for its computations starting from the focussed element; (c) the output consists of modifications in the available knowledge, a new focussed element, and an output increment.

There are a number of important points about this definition. Firstly, remember that the representations used by an incremental algorithm – focussed element, local context, available knowledge, and output increment – are managed by the incremental process calling the incremental algorithm. This includes storing the newly determined focussed element until the incremental algorithm is called (again) and sending the computed output increment to the subsequent process.

Secondly, an output increment can in particular be an update increment that modifies a previously generated output increment. Thus, the incremental algorithm must know what its previous output increments are. In my conception there are three ways in which this can be realised.

1. The incremental algorithm identifies the elements of the available knowledge together with the point of time at which they were generated.
2. The output increments are sent to an increment buffer (term \[5.13\]), not an increment stream. As long as they are in the buffer they are still accessible and can be updated, see below.

3. The incremental algorithm stores the output increments internally, which is the only possibility if the output increments are sent to an increment stream.

Since there is usually no feedback of subsequent components to previous ones in incremental models (see pages [103-104]), there is no way that a subsequent process can explicitly tell, eg via an acknowledgment signal, that an update was successful. Hence, only the first two methods provide a means to establish the success of an update via indirect feedback (term \[5.17\] on page [104]). In the third method the algorithm must ‘guess’ whether the subsequent process already processed the output increment that is to be updated and whether an update can still be successful.

Thirdly, an incremental algorithm operates on incomplete knowledge, because the available knowledge changes over time. In particular new information is integrated into the available knowledge. These future changes cannot be known to an incremental algorithm when it performs its computations, for which reason it only produces its results with respect to the knowledge available at that point. Thus, *incomplete* in this context does not mean that the representations the algorithm operates upon are not well-defined. (That would hardly make sense.) Incomplete means that not all information necessary for a complete and correct computation of an output (increment) is available.

Fourthly, I use the term incremental algorithm for naming a class of algorithms, not a particular one as Reiter and Dale, for example, for their algorithm for generating referring expressions (Dale & Reiter 1995, Reiter & Dale 1992, 2000).

Finally but most importantly, the computations take place with regard to the current state of the available knowledge. The important point is that these representations must last longer than a call of the incremental algorithm, because as I have already pointed out at the beginning of this chapter this is what incremental processing is all about. Thus, this is a crucial condition for an algorithm to be an incremental algorithm.

The notion I presented so far are the constituents of incremental models. However, before I describe how these are combined in an incremental model I introduce a useful alternative to increment streams: increment buffers.

**Term 5.13** Increment buffer. An increment buffer is a buffer between two or more incremental processes. It can store a limited number of increments that either cannot be processed further at the moment, because the reading process is not yet ready to take an increment from the buffer as input increment.

The main reason for introducing increment buffers is that for most incremental models a limited buffering mechanism is necessary between incremental processes. For most models it cannot be assumed that an incremental process is ready to process an input increment when it arrives. This function can also be fulfilled by in-
crement streams. Whether to use increment streams or increment buffers between two incremental processes is a design decision for each model. Often the simpler increment streams are adequate, especially if the processes connected by it are comparably fast, i.e., most of the time there are no increments in the stream. Artificially creating resource limitations by assuming a limited length for all increment streams would be inadequate from a cognitive modelling point of view. The slight buffering mechanism of increment streams only serves to smooth the information flow. Hence, if the unlimited size of the stream distorts the model in that substantial numbers of increments are temporarily stored between incremental processes then the limitations of storage capacity must be made an explicit issue of the model.

In contrast to increment buffers the access to increment streams very restricted. While increment streams are basically unlimited FIFO-buffers, i.e., they can only be accessed at their ends, increment buffers can allow random access. Yet, each incremental process should only have a defined set of operations it can carry out on a buffer.

An additional reason for increment buffers is that according Extended Wundt’s Principle incremental processes generate output rapidly. This often has the consequence that immediately after an output increment is generated an update increment follows in order to improve the output increment. In such cases it is much more efficient not to process the increment for a latency. This latency can be seen as time span in which the certainty increases that the increment has an adequate quality. Another advantage is that the output increments may not be ordered in a way in which they should be read by the subsequent process. The reordering can take place in an increment buffer or the reading process can pick the next increment best suited to its current state. In INC the traverse buffer has this function. It is located between the processes selection, linearisation, and PVM-generation. The refOs selected for verbalisation by the selection process are inserted into the traverse buffer, linearisation can reorder them, and after a latency PVM-generation takes them out in order to generate incremental preverbal messages.

It would also be possible to allow increment buffers within incremental processes. This would be useful for models in which the processes evaluate whether waiting is likely to pay off, e.g., with respect to the quality of the output. Since this contradicts Extended Wundt’s Principle, I do not use such buffers, and increment buffers are always located between incremental processes.

Increment buffers are limited in size (storage capacity), which for cognitive models is a way to simulate resource limitations. The available knowledge on the other hand is not limited in size, which corresponds to the distinction between general memories, where elements can be stored for an unlimited time and which is unlimited in size, while more specialised buffers store elements only temporarily and are limited in size. Levelt [1989: 26] lists three such buffers relevant for language production: working memory, the syntactic buffer, and the articulatory buffer. However, in agreement with unified cognitive architectures INC’s working memory is not limited in size and, therefore, not modelled as an increment buffer, cf. part C.
A cascade consists of incremental processes that work in parallel and that are arranged in a fixed, sequential order in such a way that each process has one preceding and one succeeding process. The incremental processes of a cascade are connected by increment streams and/or increment buffers. A process reads input increments from its preceding process and sends output increments to its succeeding process. The first process of a cascade has no preceding process but reads input from the environment, and the last process has no succeeding process but sends its output to the environment.

I already introduced the idea of a cascade above. It lies at the heart of incremental processing, because it captures the special combination of sequentiality and parallelism that is so characteristic for incrementality. A cascade is close to Reiter’s (1994) pipeline, which is the idea of subsequent processes that perform their computations one after the other. The difference is that Reiter restricts his definition to sequences where the processes do not work in parallel. Instead, the processes perform their computations in the order in which they are arranged.

The definition of cascades explicitly allows that incremental processes are connected by increment streams and increment buffers. The possibility is given mainly for the case in which incremental processes exchange their increments via an increment buffer. In this case it is more efficient to let an incremental process send a notification to its successor that a change in the increment buffer took place instead of letting the subsequent process poll the buffer, which is very costly in terms of run-time. In \textsc{inc} the traverse buffer connecting and coordinating the processes selection, linearisation, and \texttt{pvm}-generation is a case of a combined use of increment buffers and increment streams.

An incremental model is a model that contains a cascade of incremental processes and a representation of the model knowledge. The incremental model manages (a) the views of the incremental processes on the model knowledge and (b) the access of the incremental processes to the model knowledge so that only one process has access to the representation at a given point in time.

A cascade or a pipeline is generally proposed for models that process information incrementally. The idea of encapsulating this in a model and unifying the knowledge representation of the single processes into one representation, called shared memory, is a bit unusual, though. (In \textsc{inc} this shared memory is the \texttt{ccr}.) The first main reason for this is that it saves storage capacity and is closer to (psychological) memory models that typically consider human memory to be one integrated structure. However, equivalent models can be constructed that consist only of a cascade without a shared memory. In such models the knowledge is local to the incremental processes and substantial parts of the knowledge will be stored in multiple processes.

The second main reason for a shared memory is that it allows indirect feedback. Incremental models are usually strictly unidirectional, allowing no feedback.
Although this is desirable, because it makes models simple, efficient, and fast, it is not always adequate, because decisions made early on have to be carried through, before errors can be detected via a monitoring component. Therefore, it can pay off to make a process dependent on changes in the shared memory made by a later process. The discussion in chapter 5 showed that the components of the language production system are not strictly informationally encapsulated, which is support for using indirect feedback – at least for cognitively adequate models. However, this dependency must be strictly limited; most importantly, the operations of the dependent (preceding) process must not rely on feedback being available, ie indirect feedback must be optional. More on indirect feedback on pages 103–104.

The view of an incremental process on the model knowledge that constitutes the available knowledge of this process is generated and managed by the incremental model. Additionally, it must be ascertained that only one process has writing access to the shared memory at a given point of time. Otherwise the typical synchronisation problems of parallel processing arise, eg racing (‘last writer wins’). Yet, while this is a crucial issue in building artificial machines like computers, this is not an urgent issue for cognitive models. The main reason for this is that computers must be determinisitic devices in order to function properly, eg computers are usually not equipped with mechanisms for handling inconsistent representations, which are a typical result of racing. More on determinism on pages 107–108.

Since these problems are of no interest for cognitive models, I impose the constraint that only one writing process has access to the shared memory. The only reason is that it makes building and implementing the model easier without losing anything essentially required for a cognitively adequate model.

These are all notions required to build an incremental model, which can be transferred easily to incrementally operating systems and agents. The resulting behaviour of a model with the described architecture is the following.

**Term 5.16** Incremental behaviour. The minimal condition for calling the behaviour of a model incremental is that it is capable of producing output before it received all input possibly relevant for the correct and complete computation of the corresponding output. The strong condition for incremental behaviour is that additionally input and output are read/written in parallel.

The minimal condition captures what I called the processing of incomplete knowledge above. This behaviour is adequate for models/systems that interact with a human user and process the user’s input; examples are, again, compilers and text editors, cf also the discussion of partial incrementality on pages 86–87. The strong condition is a requirement for cognitive systems, eg for an agent navigating through space or for INC. This condition characterises behaviour as undesirable in which such a navigating agent (a robot, for example) scans its environments and then does not move while it evaluates this input before initiating the next movements. If INC did not meet the strong condition it could not generate online description of events. Thus, incremental models that interact with an environment with the goal
of being fluent (navigate fluently, generate language fluently) must also fulfil the
strong condition. If the processes are simple and fast, sequential (pipeline) process-
ing can be acceptable – as long as there are no observable hesitations or gaps in the
behaviour of the model.

Each model that has a cascade of more than one process is in principle capable
of bringing about incremental behaviour in the strong sense. It need not, though,
because it can be regarded as special case of a cascade if the processes are executed se-
quentially (the 'pipeline case') – if the degenerated one. There are architectures dif-
ferent from cascades that are able to behave incrementally as well, eg (a cascade of)
just one incremental process can already behave incrementally in the minimal sense.
Yet, in order to achieve strong incremental behaviour a cascade of at least two incre-
mental processes is required. A one-process model could only achieve incremental
behaviour if it were internally structured in a way that it has sub-components for
reading input and generating output simultaneously. But that would just be using
different names for the same underlying idea.

What is not explicitly required by the definition above is that in order to have
incremental behaviour proper an incremental model must indeed behave incremen-
tally if it has the capability to do so. It may behave differently in certain situations, eg
not producing output when it considers the output as not reliable enough or when
the output is not yet needed. Note, again, that this contradicts Extended Wundt's
Principle. So, if an incremental model always behaves non-incrementally although
it has the capability to do so, it would hardly be justified to say that it exhibits incre-
mental behaviour – unless one considers such behaviour as the degenerate case of
incremental behaviour.

5.4 Dimensions of incrementality

In section 5.2 we saw that incrementality is not a single notion but encompasses a
whole range of different phenomena. In the previous section I presented a blueprint
for constructing an incremental model. This is at the same time the skeleton of the
 Incrementality architecture. Yet, this skeleton does not only describe the particular architec-
ture of one model but a class of models. In this section I will describe the dimen-
sions along which these models can vary. I will do this with special consideration of
the issues relevant for conceptualisation. Hence, while describing the dimensions
along which incrementality can vary I will discuss the variants that are suitable for
conceptualisation and, accordingly, used in Incrementality.

Means of structuring. As I already pointed out in section 5.3, processes are only
one way to structure an incremental model, and especially agents are an interesting
alternative. However, agents will hardly be arranged in a cascade, because they
are used for building flexible models in which the agents depend onto each other
to a much lesser degree than processes do. Furthermore, agents typically model in-
dependent, autonomous (sub-)systems acting on their own behalf and interacting
with each other and with the environment. Consequently, the fixed, unidirectional flow of information used in cascaded models makes little sense, so that using agents as means of structuring would mean giving up the use of a cascade.

A more interesting possibility for using incremental agents is to take multiple incremental models, each one being an agent, and let them interact with each other. Then, the input to an incremental agent is generated by other agents and/or the environment, on which the agents can also act. For a conceptualiser model, however, agents offer no advantages compared with the approach taken here.

Instead of incremental models one could also use incremental programs or systems as means of structuring. Functions, procedures, or tasks could then be used instead of incremental processes. However, this would shift the focus of the proposed notions not as much as the use of incremental agents.

Monotonicity. Incremental processing can proceed monotonic and non-monotonic. In the non-monotonic case an incremental process has the possibility to generate update increments if a previously computed result was wrong or not as accurate as it should. Monotonic incrementality is, therefore, only useful in cases where no internal recomputations take place, ie in cases where the processes have no anytime characteristic at all but always produce the final result that will not be changed afterwards, eg spell-checkers. The incremental processes used in \textsc{inC} can generate update increments, ie they are non-monotonic, which enhances its robustness and of incremental models in general, cf pages \pageref{incremental_models}–\pageref{incremental_models}. Increment buffer are a good way to reduce the amount of changes in the non-monotonic case, because the incremental processes can change output increments that they stored in an increment buffer. In \textsc{inC} this is in particular the traverse buffer.

Human can change (parts of) a planned utterance when the utterance is already further down the language production system or even partly uttered. If a self-correction is possible before articulation takes place a covert correction is performed, an overt correction otherwise. Since the three main components of Levelt's (1989) language production model are not capable of giving feedback, cf section \ref{section:levelt}, a covert correction can only be performed in two cases,

1. if the output increment is still available in an increment buffer, or
2. if one of the subsequent processes receives the update increment early enough to perform the change before this process produces the corresponding output.

However, as already laid out on page \pageref{covert_correction} due to the lack of feedback the preceding process cannot know whether the update increment reaches the subsequent process in time in the second case. An overt correction can be initiated if a covert correction did not succeed or if the error was detected by the monitor component, not by the process itself.

The increments used in self-corrections are no update increments but regular output increments, because the previously generated increments are not changed. Instead, the utterance is extended, usually by a correction term like \textit{uh no}, or \textit{er}
followed by the correct content, eg

(2) *Flug CK-314 bewegt sich auf Gate B21 zu ... äh nein ... auf Gate B23.*

‘Flight CK-314 moves towards gate B21 ... uh no ... gate B23.’

INNc generates corrections like these as covert corrections, cf section 5.5. Overt corrections can be generated by INNc’s rudimentary monitor component, cf chapter 15. Yet, since there is an infinite number of possibilities to perform self-corrections, I will not explore this issue further.

Buffering. As I already pointed out, buffering is a means to make incremental models more robust and more efficient. For example, if the output increments of an incremental process are stored in an increment buffer, it reduces the run-time of the reading process to allow the writing process to change the increment in the increment buffer, before the reading process fetches it. The reason is that the reading process makes no computations with the old version of the increment. The writing process must distinguish two cases:

1. if the wrong increment is still in the buffer, it can directly apply the changes,
2. if the increment is not in the buffer any more, then an update increment must be stored in the buffer in order to be processed by the reading process.

Note that this is a case of indirect feedback, cf term 5.17 on page 104. Incremental models that use no buffering must either take special precautions that no increments get lost between incremental processes or they must be able to function properly despite the (occasional) loss of increments. In such models the processes are are typically fast and only few increments are transmitted between the processes.

Buffering is, for example, useful for handling the phenomenon of ‘word order scrambling as a consequence of incremental sentence production’ as it is described by Kempen & Harbusch [in print]. Their model of an incremental formulator obtains the parts of an utterance increment by increment. This sequence need not be in the order in which they appear in the generated sentence. The method they use is to generate the increments that can be generated in agreement with the syntactic restrictions of the language and to temporarily store the increments that would lead to a syntactically erroneous utterance. (They do not call it increment buffer, but the idea is the same.) For example, if the information that will become the verb arrives before the information that will become the subject, a German or an English formulator will have to delay the generation of the verb until the subject is generated if the utterance is a declarative sentence. Nevertheless, the delayed information can be processed to some degree, eg the lexical access for the verb can take place. INNc’s traverse buffer fulfills a similar function.

Lookahead. Lookahead makes incremental processing easier and more reliable, because lookahead means that a few subsequent input increments, which are likely to
be relevant for processing the current input increment, are available. Levelt (1989), for example, demonstrates that for some cases in language production lookahead is required. Since more input increments are known, the probability that an update increment needs to be generated straight afterwards decreases substantially. This advantage of lookahead is one of the reasons why increment buffers, in which a few increments can be collected, are a suitable way to improve the robustness of incremental models.

Yet, for incremental processing the lookahead can only be minimal: if an incremental process would wait until more than just a small number of input increments are available the rapid (and sometimes very speculative) production of output that is one basic idea of incremental processing as I put it forward here is no longer given. Thus, when designing an incremental model one has to weigh the advantages and disadvantages of using lookahead against each other.

Feedback. A crucial question for each incremental model is whether feedback between components is allowed and if so between which ones. As I repeatedly pointed out, incremental models complying to the blueprint proposed in the previous section contain no feedback. If this restriction is dropped, however, feedback in a cascade of incremental processes means that a (subsequent) incremental process \( p_s \) can send information back to a preceding process \( p_p \) in order to influence its operation, eg to request missing information or to report that an increment has been processed. While this sounds like another possibility to enhance the reliability and robustness of incremental models, it comes at a high cost, viz that an incremental process using feedback gets more complex, because it needs some knowledge about the functioning of the process(es) it sends feedback to (\( p_p \)) and /or receives feedback from (\( p_s \)). This is true for \( p_p \) as well as for \( p_s \), because \( p_s \) has to generate the feedback information and \( p_p \) must be able to interpret it.

INIC is built with the goal of using as simple a cascade as possible. This includes that its incremental processes know as little as possible about their surroundings and their neighbouring processes and that they use no feedback. Within INIC’s cascade there only is what I call indirect feedback: an incremental process \( p_s \) can influence \( p_p \) by either modifying the contents of an increment buffer or by changing a part of the model knowledge that is also part of the available knowledge of both incremental processes. \( p_p \) can change an output increment it an increment buffer accessible by \( p_p \) and \( p_s \) as long as it is actually in the buffer. After \( p_p \) has taken it out it is no longer accessible to \( p_p \), which means that \( p_s \) influenced the functioning of \( p_p \) by making it impossible for \( p_s \) to modify the increment. In INIC the only increment buffer by which indirect feedback is given is the traverse buffer, cf section 9.2. More generally indirect feedback is defined as:

Complexity increases regardless of whether the additional knowledge is implicit or explicit. Having more explicit knowledge means that \( p_p \) and \( p_s \) contain rules and inference mechanisms about how the other process functions. Having more implicit knowledge means that \( p_s \) sends back feedback as part of its normal output, eg when it sends output to its successor it may at the same time send an acknowledgment to \( p_p \). However, \( p_p \) must process this information as well.
Indirect feedback. Indirect feedback is feedback that is not realised as direct transmission of information. Instead, the component giving feedback alters a representation that the component receiving feedback is using as well. If the modification of the representation by the one component affects the operations of the other component an indirect feedback has been given.

This minimal feedback principle can hardly be kept up if incremental agents are used for modelling, because, as I already argued at the beginning of this section, agents are only a suitable means of modelling if they can interact comparatively freely with each other and if the flow of information is not as fixed as in a cascade. Hence, there are multiple information flows between the single agents, and some of the information is feedback. For example, if agent $a_1$ generated output to which agent $a_2$ reacts by generating output to which, then, $a_1$ reacts this must be considered feedback. In dialogues such feedback consists of signals from the hearer to the speaker, eg by nodding or facial expressions.

Number of increments. Incrementally working models like IPF (De Smedt 1990b) or Performance Grammar (Kempen & Harbusch 2002, in print, Kempen forthcoming) use a technique not yet introduced here explicitly: they process multiple increments in parallel – input increments as well as candidates for output increments. Thus, such systems possess an additional dimension of parallelism and additional mechanisms apart from the standard cycle of reading, processing, and generating increments, which coordinate and evaluate the increments.

Incremental processing as it is laid out here makes possible such systems, eg by providing increment buffers in which multiple increments can be stored. This captures the fact that only a limited number of increments can be processed in parallel by an incremental process because of the resource limitations. Models not concerned with limitations of resources may, of course, compute all possibilities. This leads back to the main motivation for using incrementality: a sentence or an utterance is not planned completely beforehand but piecemeal due to the limited cognitive resources of humans.

Discreteness. Computational models can be discrete or continuous. While models based on symbols are discrete, because there is no such thing as a continuous symbol, connectionist models are continuous. (One of the main tenets of connectionism is the refusal of discrete symbols after all.) Connectionist models can nevertheless in a sense be regarded as incremental models if the change of activation occurring in one place and ‘rippling’ through the network is regarded similar to the idea of piecemeal processing – except that there are no pieces, of course. For such approaches the notions I use here do not apply. Thus, this is a case where the characterisation of incrementality provided here cannot be used.

Nevertheless, the distinction between discrete and continuous models is not always as clear cut as this. Sometimes it is just a question of the perspective taken and of the properties of the model that are taken into account. For instance, network
models that use nodes to represent symbols as well as the spreading of activation for performing the computations are continuous models on the one hand – because of the spreading of activation – and discrete models on the other hand – because of the use of symbols –. One such approach is the Word Grammar by [Hudson 2003a].

Since INC is based on symbols, it is discrete without doubt. Yet, it also uses a (very limited) kind of activation to represent the degree to which the elements of representations are focussed, which influences the likelihood of them being used by processes. So, there are some continuous aspects as well, cf section 3.2.

Robustness. Incremental models are more error-prone than other models, because they produce incomplete output on incomplete input. Nevertheless, they should be as robust as possible. Thus, they should contain error correction mechanisms to improve their overall robustness and reliability, which leads to a better overall performance of the entire model. As I already argued, buffering, lookahead, and feedback are means to enhance the robustness of an incremental model, ie methods to reduce the number of errors. For parsing [Foth, Menzel, Pop, & Schröder 2000a] report on an experiment in which they used methods that allow to parse language robustly. They also show that the incremental version of their grammar is more efficient in most cases than the non-incremental version. So, once again an incremental approach to processing language saves run-time.

Strictly speaking, it is only correct to talk of robustness in relation to something, eg robustness with respect to unexpected or ill-formed input or robustness with respect to external temporal pressure [Foth, Schröder, & Menzel 2000b]. Nevertheless, one can also speak of robustness in absolute terms in the sense that something is robust with regard to normal circumstances. In this respect the notion of robustness is similar to efficiency, because one can talk of an efficient process, even though, strictly speaking it is only correct to say something is more/less efficient than something else. However, while efficiency is related to the temporal dimension only, robustness relates to multiple dimensions.

There are four possibilities when an error arises in an incremental model:

1. the error is detected but not corrected
2. the error is detected and correction is tried but fails
3. the error is detected and correction succeeds
4. the error is not detected

Each time an incremental process detects an error it must judge whether the error is severe enough to justify a correction. The decision not to correct an error may only make things worse. However, (small) errors should not affect the overall per-

* A particularly interesting result is that incrementality especially pays off in the parsing of longer sentences as opposed to shorter ones. In longer sentences the run-time of the parses is only a tenth of the run-time of the non-incremental ones, while in shorter sentences the run-time is even longer than that of the non-incremental parses.
formance of the system, because, as was already indicated, it is sensible to design an incremental model in such a way that it can tolerate erroneous input and output up to a certain degree. If the output generated up to that point is too flawed to be corrected by modifications, it (or its flawed parts) must be discarded.

There are two possibilities for corrections: update increments and further output increments. Which one to use depends, firstly, on whether it is possible to make a modification and, secondly, on whether the flawed output increments are still reachable by the incremental process. If an error is not detected it is quite likely that some even more cumbersome corrections will follow. This may well lead to a situation in which a correction is not possible and the system has to start anew.

There are different sources for errors: structural errors in the system, a malfunction of the system (performance errors), or a change that arose due to information that was read in after the output was generated. While the first error source can hardly be compensated by system internal mechanisms, the last two should be handled by incremental models. Corrections are discussed in more detail in the context of monitoring in chapter 15. For conceptualisation conceptual changes are a particularly interesting example of the third kind of errors. In conceptual changes the conceptual representation changes while an incremental preverbal message is generated for the content that is just changing.

The human language faculty is very robust, i.e., it continues processing as long as the errors are not too grave. Typical example in which the language production system continues production despite small errors are minor mispronunciations or the use of a concept that is not optimal in the current context, e.g., using the concept object instead of plane in the example of chapter 1. Although, self-monitoring is the most important means for detecting errors (Levelt 1989), they should be detected and corrected internally as far as possible, i.e., before the erroneous (part of an) utterance leaves the system. This saves the effort of an overt correction (Levelt 1983). In language production a correction may not suffice and clarifications in cooperation with the interlocutors must be performed; the error itself may become the topic of the conversation until the communicative intention is conveyed.

**Parallelism.** Although incremental models with only a single incremental process can exhibit minimal incremental behaviour, they usually contain a cascade of incremental processes. A cascade (term 5.14) is a special combination of parallelism and sequentiality. Hence, incremental models like nC that use a cascade have to deal with the standard problems of parallelism, the most important of which are:

- cooperation
- competition (for resources)
- scheduling†

† One possibility to handle even such errors is to have multiple systems that check each others’ results, a technique common in safety critical systems like airplanes.

† Scheduling is part of the competition for resources, especially if the the parallelism is only simulated.
Since there is a huge amount of literature on these topics, eg [Bic & Shaw 1988], [Herrtwich & Hommel 1994], [Tannenbaum 1987], I will only comment on some issues that are relevant for IN.C. Incremental models can vary in all these kinds, eg whether they use a synchronisation mechanism – for instance by sending signals or using semaphores – or whether they are not synchronised.

Parallelism is usually distinguished from concurrency, and sequentiality. In sequential models the sequence of all operations is fixed and cannot be changed. Parallel models use more than one process and more than one processor that work on a problem at the same time. The operating systems that run on most current computers are concurrent or quasi-parallel systems, ie they execute multiple processes, but since most computers just have one physical processor, the operating system has to bring the tasks into a sequence and has to grant and manage access to the resources, eg the central processor. IN.C is a parallel model that is implemented as a concurrent system. Since scheduling is not part of its implementation it is a close enough approximation to real parallelism.

A further kind of parallelism can be employed in incremental models, viz the use of multiple cascades that work in parallel. In models containing multiple cascades, eg one for language comprehension, one for language production, one for visual cognition, etc, the coordination problems double, because there are two interdependent kinds of parallelism: the parallelism of the incremental processes and the parallelism between the cascades. The more these systems are seen as (part of) the central executive, the more such interacting cascades exist.

Determinism. Computational models are either deterministic or indeterministic. In a deterministic model the next step of the computation – or the next state of the system – is unequivocally determined at each point of time, ie each computation, given the same input and the same state of the system, always yields the same result. Determinism is a special case of indeterminism, in which this condition does not hold. Hence, the mapping of input to output is a function in the deterministic case and a relation in the indeterministic case.
Marcus (1980) introduces an additional notion: strict determinism and demonstrates its usefulness for parsing. It differs in that the computations are no hidden indeterministic ones, ie no backtracking is used. Although in backtracking each single step is in fact deterministic the indeterminism, after backtracking took place the computation is back at a point in the solution tree it was already before. Thus, it has the same choice again but chooses a different option this time (Marcus 1980: 12). One could also call this a temporarily discontinuous, indeterministic decision.

In the context of incrementality there is a fourth notion that is useful, quasi-indeterminism. These are indeterministic computations that are performed on a deterministic device, eg a computer processor, which is deterministic but can be programmed to simulate indeterminism. Quasi-indeterminism is equivalent to the notion of concurrency introduced above.

Incremental processing as I put it forward usually means the loss of determinism due to the parallelism in the cascade. \textsc{iNC} is an indeterministic model and, as already said before, simulated parallelism (concurrency, quasi-indeterminism) is used in its implementation. Since the indeterminism is simulated by the programming system in which \textsc{iNC} is implemented, it can be taken for granted.

Consistency. As I already demonstrated in section 5.3, performing computations only with respect to a local context can mean the loss of consistency. The reason for this is that only part of the model knowledge is considered for the computation of new results, which then become part of the knowledge. Only if it can be ascertained that all relevant knowledge for a computation is part of the local context the representation stays consistent, eg the spell-checker of a word processor has to check only modified words, say if a ‘p’ is inserted in the middle of apple the spell-checker will detect that apple is now a correctly spelled word, and it can be certain that this local change does not affect the correctness or incorrectness of the other words in the text, ie the representation (which words are spelled correctly) stays consistent. However, in order to do so the frame problem must be solved, cf section 7.4. Since this is a very hard problem – apart from simple cases like the one just laid out – consistency can often only be achieved if local context and available knowledge (or even model knowledge) are identical. Yet, in some cases inconsistencies can be avoided without considering all available knowledge: firstly, if all operations carried out on the representation are conclusions in the formal sense or, secondly, if there are no changes in the environment that are not caused by the system (agent) itself.

Since I use local contexts for processing incremental representations, it is hardly possible to demand consistent representations. This is even more support for using referential nets, because they do not require consistency, especially from the perspective of cognitive adequacy, because realistic cognitive representations are inconsistent. For cognitively adequate models the inconsistencies in the model should be as close to empirically observable inconsistencies as possible. Nevertheless, I will not consider inconsistencies in the following, because the scenes \textsc{iNC} can process are not yet complex enough to warrant their assumption. Hence, the conceptual representations used here are consistent.
Modularity. I already dealt extensively with the issue of modularity in chapter 2. Therefore, I want to make just one point here: non-modular models can also work incrementality, although the way it was presented in section 5.3 implicitly assumes a modular approach and would have to be adapted accordingly.

Modular approaches stand mainly in opposition to those using network representations in which computations are carried out by the spreading of activation. The most prominent examples are connectionist networks, but there are also models that use a combination of symbols and spreading activation, e.g., Hudson claims that ‘it is all one giant network’ (personal communication, see also Hudson 2003b) and, therefore, there are no boundaries that could be identified as interfaces between modules. As already mentioned in the paragraph on discreteness such activation-based networks can be seen as performing their computations in an incremental fashion in that a change of activation of one place ‘ripples’ (‘piecemeal’) through the net. Additionally, when comparing modular and non-modular models one can observe that in modular models the structure of a model – the modularisation – is done by the creator of the model, while non-modular models very often are based on the assumption that the model possesses self-organising capabilities. Thus, self-organisation would have to be incorporated into a notion of non-modular incrementality.

5.5 Incremental representations

The previous sections of this chapter were exclusively devoted to the incremental processing mechanism. However, processing can only be performed in an incremental fashion with suitable representations. Therefore, I want to make some remarks on the representations required for incremental processing. It will come as no surprise that referential nets are the representations I have in mind, cf also chapter 4. The formalisation of referential nets will follow in the next chapter.

However, referential nets are not the only formalism suited for representations that are processed incrementally. Therefore, I want to describe underspecification formalisms as another possibility. In particular I present in this section how underspecified semantic representations in the Constraint Language for Lambda Structures (c.l.l.s., Egg, Koller, & Niehren 2001) can be generated incrementally on the level of preverbal messages.

Requirements for incremental representations. There are two important requirements a representation must fulfil in order to be used for incremental processing. Firstly, it must be changeable over time, and, secondly, it must allow to define increments. For the way I define incremental processing the representational formalism must, furthermore, represent knowledge in a localised fashion, which is required for the computation of local contexts, and it must support different views on the representation for each incremental process.

A representational formalism must possess a third set of requirements in order
to be used for conceptualisation. Firstly, it must be able to represent (hierarchical) conceptual knowledge. Secondly, it must provide means for linking the conceptual knowledge to non-conceptual knowledge, ie perceptual and linguistic (semantic) knowledge, cf the remark on the symbol grounding problem in section 4.2. Thirdly, it must represent knowledge in a cognitively adequate manner, eg it must allow to represent inconsistencies.

The discussions up to now already showed that referential nets fulfil these properties. In comparison, CLS only fulfils the first, general set of requirements (changeability, definition of increments) to its full extent, the second (localisation, different views) only with some additional definitions, and does not possess two of the three requirements of the third kind (representing conceptual knowledge, linkage to non-conceptual knowledge), while it possesses certainly some cognitive adequacy.

Two kinds of knowledge representations are needed in INC, conceptual representations and incremental preverbal messages – which are really just special conceptual representations. Both are built up by assembling the representation piece-meal. The changes in the representation can be subdivided into adding or deleting refOs and designations and adding, deleting, and modifying information associated with a refO or a designation, ie attributes, cf the next chapter.

Referential nets as I use them here are extended compared to the standard version (Habel 1986, Eschenbach 1988) by a rudimentary activation concept, cf chapter 6. Each element of a representation in INC has an activation value, which determines its prominence (salience) in comparison to the other elements. Currently, the activation in INC is static, ie an activation value is assigned once and then never changed. In order to enhance its cognitive adequacy mechanisms for the decay and reactivation of items must be integrated, cf Anderson & Lebiere (1998).

With regard to the dimensions discussed in the previous section four are also relevant for characterising incremental representations. The representations used here are non-monotonic, ie elements can be changed by update increments, and they can be deleted. The output increments of an incremental process can be buffered before they are consumed as input increments of another incremental process. Since referential nets are a symbolic representational formalism, representations are discrete. Finally referential nets allow inconsistent representations.

Underspecification and incrementality. Semantic underspecification formalisms have been proving their usefulness over the last years. There are two main reasons for this. The first reason is that underspecification makes it possible to represent ambiguous utterances by only one structure for all readings instead of one structure for each reading, in particular in the case of scope ambiguities. At the same time it is an elegant representational method for the semantic description of anaphora,
reinterpretation in lexical semantics, and the meaning of elliptical expressions (Egg, Koller, & Niehren 2001, Schilder & Guhe 2002). Although the idea of semantic underspecification is not particular to one direction of language processing, it has – with few exceptions – only been used in comprehension up to now. This probably is so, because the mentioned problems are typical problems in the parsing of sentences and discourses, while they can be more or less avoided in generation. The problem that particularly sets aside comprehension from generation is that during parsing ambiguities arise that usually can only be resolved by considering the corresponding context. Most parsers, though, work without a context, because their goal is to build up a semantic representation, which is – by and large – independent of context. Since one of the main advantages of underspecification is to represent ambiguous utterances in one structure, only one underspecified semantic structure has to be built up during parsing which can then be disambiguated in a further step – if there is need at all, cf. Knight & Langkilde (2000), who describe a system that preserves ambiguities during translation.

However, these parsing problems have no real counterparts in NLG and language production, cf. section 2.1. The main reason is that the knowledge representation from which language is generated usually is completely available. For example, when I use the name Tom I always know which Tom I am talking about even if I know more than one person with this name. Yet, a person or a system hearing me say Tom will have to infer which of the, say, seven Toms known to us both is meant. In the case of anaphora the generator always knows what the anaphora is referring to; for an ellipsis the implicit content is known, and since it is known what is meant when the lexical semantics is generated there is no need for reinterpretations. Most builders of NLG systems even explicitly try to avoid these problems if they arise while specifying a system. However, if the goal is to build a cognitively adequate system like nC, leaving information implicit is a very desirable property of the representational formalism. This can also be done in referential nets, where it is called underdetermination (‘Unterbestimmtheit’, Habel 1986), which has additional, different connotations. For example, the cardinality of a refO representing a group of entities can be underdetermined by stating that the number of entities the group consists of is between, say, three and twelve; the exact number is not specified.

The second and in this context the important reason for using underspecification is that underspecification can be used for representations that change over time. An underspecified representation is usually not totally underspecified, but in some parts it is specified. It is, therefore, possible to distinguish those parts of the representation that cannot be modified (without creating an error or reanalysis) from those parts where elements can be added or inserted. It is this extendability property of underspecification that can be exploited for the incremental generation of preverbal messages.

While underspecified structures have been used in NLG before, it was never suggested to exploit the extendability property of underspecification formalisms. For example, Pianta & Toven (1999) present a hybrid generation system in which
In the resulting system the information to be verbalised can be specified on different levels, i.e., the underlying data structure can contain a string that is output as is as well as other knowledge representations for which output must be generated by other NLG techniques. However, this is more of a hybrid system instead of one that systematically uses underspecified representations. The Verbmobil system, a machine translation system [Wahlster 2000] that uses a semantic transfer approach, comes closest to the idea put forward here. It uses underspecification in the semantic representations and components for parsing and generation that work incrementally. However, it uses a dialogue-act based translation, which means that each dialogue act is completely parsed into an underspecified semantic representation and then completely translated before generation commences. In this way the advantages of underspecification mentioned above – plus, to some extent, the preservation of lexical ambiguities – can be used but not the fact that an utterance can be generated incrementally while planning is not finished. Finally, [Gardent & Thater 2001] propose to use underspecification on the level of the formulator. A modified output of inC could in fact be used as input to this system.

As I already argued in part A, incrementality is a way to cope with dynamically changing states of affairs, e.g., for models working in a dynamic setting. The dynamics of the environment and incremental processing necessitate changes in the representation of the states of affairs. The main advantage of underspecification is that these representations need not be restructured or constructed from scratch after each change. Quite the contrary: like in the referential nets approach new information can be added to the already existing underspecified representation at the allowed places. One can imagine these places as holes in the representation where new information can be filled in. However, this is not obligatory; the representation is already complete as it is. This method facilitates models that need not plan a whole utterance in advance to generate a valid structure but that can plan an utterance piecemeal while it is already leaving the system.

Summing up, underspecification can be employed for incremental language generation, because an underspecified representation can be extended without reverting earlier made planning decisions. Subsequently planned increments can be inserted where the underspecified representation left room. Since underspecification is not mainly motivated by cognitive considerations, an additional aim of this section is to show that NLG techniques can be enhanced by transferring work from cognitively oriented language production to NLG in the sense of [Reiter 1994].

Generating incremental preverbal messages with CLLS. The Constraint Language for Lambda Structures (CLLS, Egg, Koller, & Niehren 2001) is a formalism for the partial (underspecified) description of lambda structures. Lambda structures are represented as ordinary trees amended by the two partial functions lam for binding variables of the λ-term and ante for modelling anaphoric expressions. The lambda

* This view assumes that template-based generation is no NLG technique.
term $\text{Mary}(\lambda x.\text{sleep}(x))$, for instance, can be represented as the tree structure shown in figure 5.5. The label @ indicates application, a dashed line between two nodes labelled by lam and var ensure the correct binding between variables and the $\lambda$-abstractor. Additionally, variables denoting tree nodes (e.g. $X_0, X_1, X_2, \ldots$) are added to the lambda tree structure in order to allow for underspecification, the specification of anaphoric references, etc. Formally, a clls formula is described as a conjunction of atomic literals. In order to satisfy such a formula a lambda tree structure, eg the one in figure 5.5, and variable assignments have to be found such that every literal is satisfied.

Several constraints are defined in clls. Crucial for the definition of underspecification is the dominance relation holding between tree nodes: $X \prec^* Y$ is satisfied if and only if $X$ denotes an ancestor of $Y$ in the lambda tree structure. The constraint graph indicates this relation via the dotted lines, eg the relation between the nodes labelled by $Y_1$ and $Y_2$ in figure 5.6. The dominance relation is reflexive and transitive; thus, the nodes connected via a dotted line can be identical or there can be an infinite number of further tree structures inserted between the nodes. Another constraint ensures that the binding between variables and lambda operators is given: $\lambda(X) = Y$ is satisfied if and only if the denotation of $X$ maps to the denotation of $Y$. Within the constraint graph the mapping is indicated by the dashed line pointing from the variable to the lambda operator (lam).

For the representation of coordination-structures the parallelism constraint is particularly interesting. It defines a parallel structure between tree segments. Segments in a lambda structure are defined as $X/Y$ where $X$ denotes the root of the segment and $Y$ a hole such that $X \prec^* Y$. The segment covers all nodes that are dominated by the root $X$ with the exception of the node $Y$ and the nodes dominated by $Y$. In other words, a segment is a sub-tree starting with the node $X$ with the exception of a further subtree, which has $Y$ as root node. For instance, in figure 5.6, the segment $X_1/X_2$ has the root node $X_1$ including all nodes dominated by it apart from node $X_2$ and the subtree of which $X_2$ is the root. The actual parallelism constraint $X_1/X_2 \sim Y_1/Y_2$ is satisfied if and only if the segment $X_1/X_2$ of the lambda structure is parallel to a segment $Y_1/Y_2$. The segments are described by brackets in the

\footnote{NPS including PNS are type-raised. Hence, the term $\text{Mary}$ in $\text{Mary}(\lambda x.\text{sleep}(x))$ is a function from sets of entities to truth values.}
constraint graphs (see figure 5.6). Formally, the parallelism between two segments is captured via a correspondence function which is defined as a bijective mapping between the two segments, cf. Erk (2000) for further details.

The parallelism constraint proves to be especially useful for the description of \( \lambda \) ellipses and for self-corrections, like in the following examples:

(3) Mary sleeps and John does, too.
(4) Mary sleeps . . . uh no . . . is awake.

The corresponding CLLS representation for (3) is given in figure 5.6, the one for (4) in figure 5.7. Note that the parallelism constraint for the tree in figure 5.7 is not written in the bracket notation but as constraint below the tree.

The advantage of CLLS and underspecification formalisms in general is that semantic representations for utterances like these need not be generated at once. In the first stage of generating the above examples the representation in figure 5.3 is produced; or, to be more precise, this structure is generated in its underspecified version shown in the left side of the tree in figure 5.6. This is done in two steps. Firstly, the tree containing the underspecification for \( \lambda x.\text{sleep}(x) \) is produced, secondly the node for Mary and the application (@) of the lam tree to the Mary node. This tree can now be extended at two points, above the upper @ node and between \( X_4 \) and the lower @. The latter can be used for utterances like:

(5) Mary sleeps and snores.

The former possibility is used in the examples presented here: \( \lambda \) ellipses and self-corrections. After the generation of the initial tree the conceptualiser may detect that not only Mary is sleeping but John as well, or it may detect that Mary in fact is not sleeping in contrast to its former belief. In the first case a \( \lambda \) ellipsis can elegantly
extend the utterance generated so far, in the second case the erroneous part of the utterance can be corrected. Note that in the case of a change of belief it would be misleading to simply generate an utterance that reflects the new current state.

(6) Mary sleeps, Mary is awake.

This utterance must be regarded as a faulty description of the state of affairs, because it confusing the hearer with respect to which of the two utterance is actually true. In general, it cannot be expected that the hearer simply overwrites former beliefs. Instead, it must be made explicit that part of the previous utterance is wrong.

The parallelism constraint $X_1/X_2 \sim Y_1/Y_2$ occurring in figure 5.6 is reflected in the graph via two brackets denoting the two parallel segments (i.e. $X_1/X_2$ and $Y_1/Y_2$). The brackets precisely determine the part of the source sentence (Mary sleeps) that has to be copied into the target sentence (John does, too) as well as the part that has to be kept separate (Mary and John). One of the lambda tree structures satisfying this constraint graph is given in figure 5.8. This specification step is tantamount to deciding upon one reading of the sentence.

A few more details are necessary in order to use this approach with $\text{inc}$, e.g., a function translating designations and refs to $\text{clls}$ terms (cf. Guhe & Schilder 2002b), which are of no interest here. The way incremental preverbal messages are generated with $\text{inc}$ in the referential nets formalism will be described in chapter 13.

Underspecification is a very powerful formalism, as can be seen by the fact that it can be taken even one step further: not only the structure but also the constraints themselves can be underspecified. In particular, the parallelism constraint can be underspecified in order to capture all three readings of the gapping example (7), given in (6), cf. also Schilder & Guhe (2002).
Figure 5.8: A lambda structure that satisfies the constraint graph in figure 5.6.

(7) Peter gave Mary a book and John, too.
    b. Peter gave Mary a book and John gave Mary a book.
    c. Peter gave Mary a book and Peter gave Mary John.

Distinguishing the first two readings is a standard problem of semantics and is, therefore, not further commented on. The third reading is usually ruled out by standard pragmatic assumptions but may be the correct reading if, for instance, John is Mary’s favourite doll. The basic idea of underspecifying the parallelism constraint is that the reading that actually fits the state of affairs is not specified, ie the attachment point of the gapping expression is not specified. In other words, it is left open which of the three argument positions of the VP formed by give the gapping construction is attached to. The discussion of the technical details required for this are given in Schilder & Guhe (2002) and are beyond the scope of this investigation, because underspecifying the parallelism constraint is mainly interesting for language comprehension. The main reason for this is that the attachment point is known to the generation system. However, it demonstrates that underspecification in this dimension, ie underspecifying a constraint, reduces the amount of information that has to be transmitted from speaker to hearer and, thus, is another means to save resources.

Concluding, one can say that CILS allows to specify very flexible and concise structures that can be modified in the course of time. For this reason they are suited for incremental processing.
FORMALISING the notions I introduced in the previous chapter will serve to make my proposal of the incremental processing mode more precise. In particular it serves as next refining step towards providing a blueprint for building incremental models. This includes especially INC. I already introduced all notions defined and described in this chapter. Hence, I will present no new ideas, and the reader not interested in this formalisation may leave out this chapter without losing track of my argument.

For the definitions in this chapter I use the formal specification language Z. I chose this language instead of, say, simple predicate logic, because it supports a step-wise refinement of the specification. This means that the following definitions can be taken as a starting point and extended in order to specify a particular system. I use this method already within this chapter. In section 6.1 the terms introduced in section 5.3 are formalised, and section 6.2 refines this formalisation by specifying referential nets, cf chapter 4. Furthermore, Z provides a lot of mathematical definitions, which simplifies the specification process, because time and space need not be spent on defining well-known notions like bijections, sequences, or recursive types. Finally, Z is rather widely known (for a specification language) and has already been used for the specification of cognitive models, eg for a specification of the Soar architecture [Milner 1992].

I will use Z as it is defined by Spivey [1992]. (For a more comprehensive introduction see Wordsworth [1992].) I will shortly describe the most important concepts of Z in the text as we go along. The formalisation in this chapter was type checked with the fuzz program.

6.1 Incremental processing

I will provide the formalisation as far as possible, because some procedural aspects cannot be modelled by Z, especially the condition that an incremental process is running in a continuous loop until it is explicitly terminated from the outside. In

fuzz is free software and can be obtained at [http://spivey.oriel.ox.ac.uk/mike/fuzz/]
Z such properties are therefore not formalised but given in the running text, which means that the text is part of a Z specification.

Basis. The basis of the formalisation is a formal language $L$ that defines the set of possible words (expressions). In the case of rNC this language will be defined as referential nets in section 6.2:

$$[L]$$

This Z notation introduces a basic set. In the specification of an actual model such a set must be defined, e.g. by enumerating all elements. 

Increments. Increments are then simply words of this language $L$.

Increment $\equiv [\text{increment} : L]$

The $\equiv$ sign is one notation for introducing new Z schemas, which are the main means to structure a specification. Increment is the name of the schema, and [...] contains the definition. In this case it can also be thought of as a type definition. So, this schema says that an Increment is a schema containing one element, called increment that is an element of the set $L$.

Schemas make it possible to present the specification piece by piece, each of which can be commented on in the accompanying text. They can describe both, static and dynamic aspects of a specification. 'The static aspects include: the states it can occupy; the invariant relationships that are maintained as the system moves from state to state. The dynamic aspects include: the operations that are possible; the relationship between their inputs and outputs; the changes of state that happen.‘ [Spivey 1992]

Update increments are a special kind of increments.

UpdateIncrement $\equiv [\text{Increment}]$

When defining a schema not only new values can be introduced but old ones can be reused as well. Expanding this definition yields

$$\text{UpdateIncrement} \equiv [\text{increment} : L]$$

The sets of input and output increments are defined analogously.

InputIncrement $\equiv [\text{Increment}]$

OutputIncrement $\equiv [\text{Increment}]$

With this the following relations hold:

InputIncrement $\subseteq$ Increment
OutputIncrement $\subseteq$ Increment
Increment =

InputIncrement $\cup$ OutputIncrement $\cup$ UpdateIncrement
UpdateIncrement $\cap$ InputIncrement $\neq \emptyset$
UpdateIncrement $\cap$ OutputIncrement $\neq \emptyset$
UpdateIncrement $\subseteq$ Increment

* The result of the fuzz type checker was obtained by ignoring this definition, because basic sets cannot be refined as I did it in section 6.4.
Mathematical expressions like these specify constraints on the model.

These constraints say that, firstly, input and output increments are only special kinds of increments. Secondly, increments can be input increments, output increments, and update increments (and nothing else). Thirdly, update increments are input increments, output increments, or both at the same time. Finally, update increments are a proper subset of the set of increments, because an update increment always needs an increment that is no update increment for which it is an update. Therefore, I also define (as an auxiliary type) the set of those increments that are 'proper' increments.

\[
\text{ProperIncrement} = \left[\text{Increment}\right] \\
\forall i : \text{ProperIncrement} \cdot i \in (\text{Increment} \setminus \text{UpdateIncrement})
\]

Quantifiers in Z are used in the form \( Q \ S \bullet E \) where \( Q \) is a quantifier, \( S \) a schema text (ie a declaration (of variables) and an optional list of predicates constraining the declaration, written \( D \mid P \)), and \( E \) the quantified expression.

An update increment always updates exactly one previous increment, ie it is applied to the increment to be updated, which is done by the function \( \text{update} \). The resulting increment is then an increment that is no update increment.

\[
\text{update} : (\text{UpdateIncrement} \times \text{ProperIncrement}) \rightarrow \text{ProperIncrement} \\
\text{updateFor} : \text{UpdateIncrement} \rightarrow \text{ProperIncrement}
\]

This is an axiomatic description by which global variables are introduced. Additionally, constraints over their values can be specified, but this is not done in the above definitions. The first axiomatic description defines the function \( \text{update} \), which is a function taking two arguments (one update increment and one proper increment; the \( \times \) is the Cartesian product) and computes a proper increment. The \( \rightarrow \) arrow indicates that it is a partial function, the \( \rightarrow \rightarrow \) arrow a total function.

Note that \( \text{update} \) is only a partial function, ie an update increment can be applied only to a suitable proper increment. This has to be ascertained in the specification of a model.

**Increment streams.** An increment stream is a sequence of increments:

\[
\text{IncrementStream} = \left[\text{stream} : \text{seq Increment}\right]
\]

Sequences are a predefined in Z. ‘\( \text{seq} \ X \) is the set of finite sequences over \( X \). These are finite functions from \( N \) to \( X \) whose domain is a segment \( 1 \ldots n \) for some natural number \( n \).’ [Spivey 1992:115]

The increment sequence contained in an IncrementStream can be given as \( \langle i_1, \ldots, i_n \rangle \). The empty sequence is written as \( \langle \rangle \).

Only two operations can be used on an increment stream. Append appends an increment at the end of the stream and Fetch gets the first increment from the increment stream while removing it. Z provides no means to prohibit the definition of further operations on the schema IncrementStream in the specification of an actual model. However, remember that a textual specification like this suffices.
Append

\[ \Delta \text{IncrementStream} \]
\[ e? : \text{Increment} \]
\[ \text{stream}' = \text{stream} \setminus \{e?\} \]

Fetch

\[ \Delta \text{IncrementStream} \]
\[ e! : \text{Increment} \]
\[ \text{stream}' = \text{tail stream} \]
\[ e! = \text{head stream} \]

This is the alternative notation for introducing new schemas. It consists 'of a part above the central dividing line, in which some variables are declared, and a part below the line which gives a relationship between the values of the variables.' (Spivey 1992:3) The schemas given up to now only introduced new variables without specifying their relations.

The notation \( \Delta S \) for an already defined schema \( S \) is used for operations on this schema. In this case it introduces the variables \( \text{Stream} \) and \( \text{Stream}' \), the first of which contains the value before the operation, while the primed version contains the value afterwards. Each pair of variables is constrained to satisfy the invariant given by the schema predicate (the part below the line). Hence, it must hold before as well as after the operation. By \( \text{sq}_1, \text{sq}_2 \) two sequences are concatenated. Using \( v? \) means that \( v \) the variable \( v \) contains an input value, the use of \( v! \) that it contains an output value. \( \text{tail} \) and \( \text{head} \) serve to decompose a sequence; \( \text{tail} \) returns the sequence without the first element, and \( \text{head} \) returns the first element.

Increment buffers. Increment buffers are based on the definition of increment streams. They differ from increment streams in that they are of limited length, ie they can store only a limited number of elements, and in that they can be accessed more flexibly, ie more operations can be defined for manipulating an increment buffer. The additional operations for manipulating an increment buffer have to be defined when a model is specified, because they are different for each buffer.

IncrementBuffer

\[ \text{IncrementStream}[\text{buffer/stream}] \]
\[ \text{maxLength} : N \]
\[ \#\text{buffer} \leq \text{maxLength} \]

Schema\( \{a/b\} \) is the definition of the schema, but the variable \( b \) is renamed to \( a \). \( \#\text{sq} \) is the length of the sequence \( \text{sq} \).

Available knowledge. Like increments, the available knowledge is also defined with respect to \( \mathcal{L} \). It is a set of words of the language \( \mathcal{L} \):

\[ \text{AvailableKnowledge} \equiv \{\text{knowledge} : \mathcal{P} \mathcal{L}\} \]

\( \mathcal{P} S \) denotes the power-set of \( S \).
The available knowledge is defined as the knowledge that is available to an incremental process. Thus, the schema View on page 127 defines the available knowledge as the process’s view on the model knowledge. There are three operations on AvailableKnowledge. Firstly, Add adds an element to the set of the available knowledge.

Add

\[
\begin{align*}
\Delta \text{AvailableKnowledge} \\
e? : \mathcal{L} \\
\text{UI} : P \text{UpdateIncrement} \\
\forall x : \text{UI} \cdot e? \neq x.\text{increment} \\
\text{knowledge}' = \text{knowledge} \cup \{e?\}
\end{align*}
\]

The notation $S.var$ selects the variable $var$ from the schema $S$.

Observe that the definition of Add allows in particular to add increments to the available knowledge. Yet, update increments must be excluded from this, because they only contain information for modifying a proper increment. $\text{UI}$ is the set of update increments, and for all $x$ that are in $\text{UI}$ it must hold that $e?$, the element to be added to the available knowledge, is not equal to such an increment. $\text{knowledge}'$, the available knowledge after the operation, is then equal to $\text{knowledge}$, the available knowledge before the operation, enlarged by $e$.

Secondly, Delete removes an element from the available knowledge.

Delete

\[
\begin{align*}
\Delta \text{AvailableKnowledge} \\
e? : \mathcal{L} \\
e? \in \text{knowledge} \\
\text{knowledge}' = \text{knowledge} \setminus \{e?\}
\end{align*}
\]

For the third operation, Update, we first need to define a function $updateWord$ that takes a word from $\mathcal{L}$ and an update increment to compute the updated word. It has to be specified for the language and the model individually.

\[
\text{updateWord} : (\mathcal{L} \times \text{UpdateIncrement}) \rightarrow \mathcal{L}
\]

$updateWord$ differs from $update$ defined above in that it not only takes increments but all words from $\mathcal{L}$, because the available knowledge can contain not only increments. A simple operation to update an element of the available knowledge is to remove the old element $e?$ from the available knowledge and to add the updated one.
For actual models it is more efficient to define optimised update operations that change only the information of the word to be changed. For referential nets this operation will in section 6.2 be defined as one that adds and/or deletes attributes and/or designations but not all information associated with a refO.

**Focussed element.** A focussed element – based on which a local context is computed – is an element of the available knowledge. Each incremental process has a focussed element, which is either the last input increment, or it was determined in the last application of the incremental algorithm of the process, see below.

\[ \text{FocussedElement} \equiv [\text{AvailableKnowledge}; \text{fe} : \mathcal{L} \mid \text{fe} \in \text{knowledge}] \]

**Local context.** The local context is an interconnected subset of available knowledge. Therefore, I first define connectedness. Two elements of a set are connected if they or their inverse are related by a relation R.

\[
\begin{align*}
[X] & \quad \text{connected}_L : P(X \leftrightarrow X) \\
\forall R : X \leftrightarrow X \bullet (\text{connected } R & \Leftrightarrow \\
& \quad (\forall a, b : X \bullet (a, b) \in R \lor (b, a) \in R))
\end{align*}
\]

Schemas with a double line introduce generic constants. These are 'mathematical constructions that are independent of the elements from which the construction starts'. (Spivey 1992: 38) Thus, X in this definition can be instantiated by any set and R by any relation fulfilling the specified constraint.

A local context (LC) is then a set of words of \( \mathcal{L} \) with respect to a Relation that ascertains the connectedness.

\[
\begin{align*}
\text{LocalContext} & \quad \text{AvailableKnowledge} \\
& \quad \text{FocussedElement} \\
& \quad \text{LC} : P(\mathcal{L}) \\
& \quad \text{Relation} : \mathcal{L} \leftrightarrow \mathcal{L} \\
\text{LC} \subseteq & \text{knowledge} \\
\text{fe} & \in \text{LC} \\
\text{connected Relation} & \\
\forall e_1, e_2 : \text{LC} \bullet (e_1, e_2) \in \text{Relation} \lor (e_2, e_1) \in \text{Relation}
\end{align*}
\]
The $X \leftrightarrow Y$ notation defines the set of binary relation between $X$ and $Y$.

There must be a connectedness relation so that for all pairs of elements $e_1, e_2$ in the local context $LC$ it holds that they are connected (in any order). Observe that the local context can be equal to the available knowledge. This is especially important if no inconsistencies are allowed or for models using full incrementality, cf chapter 5. Then this equality must hold.

**Incremental algorithm.** An incremental algorithm is a function that takes a focussed element, a local context, and the available knowledge in order to produce the new focussed element, an output increment, and the new state of the available knowledge. The definition is split into two parts, firstly, the function proper that matches the input triple to the output triple ($\text{incrementalAlgorithm}$), and secondly, the operation schema $\text{IncrementalAlgorithm}$ that links the execution of $\text{incrementalAlgorithm}$ to the schemas defined so far. Remember that update increments are just special increments. For this reason the output increment can be a proper increment as well as an update increment.

\[
\text{incrementalAlgorithm} : (L \times P L \times P L) \rightarrow (P L \times L \times \text{Increment})
\]

\[
\text{IncrementalAlgorithm} \rightarrow \Delta \text{AvailableKnowledge} \\
\text{LocalContext} \rightarrow \Delta \text{FocussedElement} \\
\text{outputIncrement}! : \text{Increment} \\
\text{knowledge}', \text{fe}', \text{outputIncrement}! = \text{incrementalAlgorithm}(\text{fe}, \text{LC}, \text{knowledge})
\]

**Incremental process.** An incremental process can be understood as a container for the call of an incremental algorithm. It has assigned two increment streams, an input stream and an output stream that establish the connection to other incremental processes, which, taken together, form a cascade, see below.† The increments that can be in these streams are the input and output increments of the particular process.

* Since changes can also occur outside the local context, an incremental algorithm can modify all available knowledge, not only the elements in the local context. In rsc this is, for example, the case when the incremental algorithm of the construction process inserts expected elements into the ccr.

† An alternative to this is to allow multiple input and output streams for an incremental process. Although this is a useful extension, for the definition of cascades it would only complicate things without adding anything substantially new. In rsc this possibility is in fact used for the construction process that interchanges information with the concept matcher via increment streams and that receives input from the rpm via another increment stream. The concept matcher and the concept storage can be seen as part of construction; then, these problems are avoided. See section 10.1 on the reasons why I did not choose this possibility.
IncrementalProcess

IncrementalAlgorithm
inputStream, outputStream : IncrementStream
inputIncrements : P InputIncrement
outputIncrements : P OutputIncrement
feP : FocussedElement

∀ i : Increment | ⟨i⟩ in inputStream.stream • i ∈ inputIncrements
∀ i : Increment | ⟨i⟩ in outputStream.stream • i ∈ outputIncrements

The notation sq₁ in sq₂ means that sq₁ is a sub-sequence of sq₂, e.g. ⟨b, c⟩ in ⟨a, b, c, d⟩.

Instead of increment streams for the input and output streams increment buffers can be used as well. However, in order not to clutter the specification with too much detail – increment buffers are simply increment streams with an additional constraint – I will not provide the corresponding Z-schemas, which are analogous to the ones provided here.

Since the Z notation provides no means to formalise notions like recursions or infinite loops, I give the schema for one run of the infinite loop that an incremental process executes in ExecuteIncrementalProcess. Remember that an incremental process is terminated explicitly. This schema captures the fact that the focussed element of an incremental process (feP) is either an input increment from the input stream or the old focussed element. The new focussed element of the process is the focussed element given back by the incremental algorithm. The output increment is appended to the output stream. This includes the case where no output increment is generated, because in this case the empty sequence ⟨⟩ is appended.

ExecuteIncrementalProcess

∆ IncrementalProcess
hi : Increment

inputStream.stream = ⟨⟩ ⇒ fe = feP.fe
inputStream.stream ≠ ⟨⟩ ⇒
  hi = head inputStream.stream ∧
  (fe = if hi ∈ UpdateIncrement then (update(hi, updateFor(hi))).increment
   else hi.increment) ∧
  (inputStream′.stream = tail inputStream.stream)
knowledge′ = knowledge ∪ {fe}
feP′.fe = fe′
outputStream′.stream =
  outputStream.stream ⊕ ⟨outputIncrement!⟩

X ⇒ Y is the Z notation for the implication. This schema uses a production-like notation to distinguish the cases when an input increment is available in the input stream from when there is none.
An incremental process works as follows. At first it is checked whether an input increment is available in its input stream. If the input stream is empty, the new focussed element is the same as before. If there is an input increment available in the input stream then it is fetched. If it is an update increment the increment for which it is an update is modified accordingly. The updated increment is then the focussed element of the process for the execution of the incremental algorithm. If the input increment is no update increment it must be a proper increment. The focussed element is added to the available knowledge. (If it is already contained in the available knowledge this has no effect.)

By schema inclusion it is also ascertained that the focussed element for the next call of the incremental algorithm is determined and the local context is computed from focussed element and available knowledge. As we saw above, the relation by which the local context is determined is not formalised here, because it is particular to the incremental algorithm carried out within the process. Note that it is easily possible to extend the formalisation by making the process capable of executing different incremental algorithms depending on further factors, eg the currently focussed element.

With the focussed element and the local context the incremental algorithm is executed, which yields a new state of the available knowledge (knowledge''), a new focussed element of the process (feP'), and an output increment. To reduce useless computations for the cases in which knowledge'' = knowledge' and/or feP' = feP one could allow that an empty output increment is not considered any further, ie it is not appended to the output stream and the process suspends itself until a new input increment is available in the input stream. This possibility is used in inc. Finally, the output increment! is appended to the outputStream.

**Cascade.** A cascade consists of a sequence of incremental processes that are linked by increment streams. To be more precise: two processes are connected by one increment stream.

\[
\text{Cascade} \quad \text{cascade : seq IncrementalProcess} \\
\text{inputStream, outputStream : IncrementStream} \\
\text{inputStream} = (\text{head cascade}).\text{inputStream} \\
\text{outputStream} = (\text{last cascade}).\text{outputStream} \\
\forall p_1, p_2 : \text{IncrementalProcess} \quad \\
\forall \text{pref, suff : seq IncrementalProcess} | \\
\quad \text{cascade} = \text{pref} \triangleleft (p_1, p_2) \triangleleft \text{suff} \quad \\
\text{(p}_1, \text{outputStream} = p_2, \text{inputStream}) \land \\
\text{(p}_1, \text{outputIncrements} = p_2, \text{inputIncrements})
\]

* This makes necessary a slight extension: incremental processes must be initialised with a focussed element. The incremental model that contains the incremental process can, therefore, only start after this assignment is made.
There are two increment streams that connect a cascade to the outside, the input stream of the first process (head cascade) and the output stream of the last (tail cascade). Two adjacent processes of the cascade are connected by an increment stream, and the output stream of the preceding process is the same stream as the input stream of the subsequent one. This also requires that the increments in this stream are output increments of the preceding and input increments of the subsequent process.

Incremental model. For the definition of the incremental model we need the definition of the model knowledge:

\[ \text{ModelKnowledge} \equiv [\text{knowledge} : P \mathcal{L}] \]

An incremental model consists of a cascade and the model knowledge. The available knowledge of the incremental processes in the cascade is a subset of the model knowledge. The input and output stream of an incremental model are the ones of the cascade, ie the input stream of the first process and the output stream of the last.

\[ \text{IncrementalModel} \]

\[ \text{ModelKnowledge} \]

\[ \text{Cascade} \]

\[ \forall p : \text{IncrementalProcess} \mid (p) \in \text{cascade} \bullet p.\text{knowledge} \subseteq \text{knowledge} \]

This definition corresponds to the definition of term \[^{5,15}\] on page \[^{98}\]. However, it can be useful to define incremental models that have processes outside the cascade. (The Z-schema would have to be adapted accordingly.) For this it is necessary that processes can have multiple input and output streams. InC has such a further process, the concept matcher, which is connected to the construction process via increment streams.

Before I define the accessibility of the model knowledge and the different views on it I first introduce an auxiliary schema IM that serves to hide the input and output stream of the incremental model, which play no role in those definitions.

\[ \text{IM} \equiv \text{IncrementalModel} \setminus (\text{inputStream}, \text{outputStream}) \]

The definition Schema \[^{(a, b, c)}\] is the definition of the schema without the variables a, b, and c.

Since the model knowledge is stored in a shared memory, the incremental model must manage the access of the incremental processes to the knowledge. This includes that a process does not know about the access restrictions imposed by the model. The first restriction is that a process has a certain view on the model knowledge, which is defined by the function \(\text{view}\).

\[ \text{view} : \text{IncrementalProcess} \times \text{ModelKnowledge} \rightarrow \text{AvailableKnowledge} \]
view is a partial function, because there may be incremental processes that do not
need access to the shared memory. The following schema View provides the cor-
responding assignments.

\[
\begin{align*}
\text{View} & \ \\
\text{model} &: \ IM \\
\text{mk} &: \ ModelKnowledge \\
\text{proc} &: \ IncrementalProcess \\
\text{ak} &: \ AvailableKnowledge \\
\langle \text{proc} \rangle \in \text{model.cascade} & \\
\text{mk.knowledge} &= \text{model.knowledge} \\
\text{ak.knowledge} &= \text{proc.knowledge} \\
\text{view}(\text{proc}, \text{mk}) &= \text{ak}
\end{align*}
\]

The second access restriction is due to the parallelism (concurrency) of the pro-
cesses accessing the model knowledge. It must be ascertained that only one process
operates on the knowledge at a given point of time. For this reason the set of all
time points is defined as basic set.

\[ \mathcal{T} \]

The function access now determines the incremental process that has access at a
given point in time.

\[
\text{access} : \mathcal{T} \rightarrow \text{IncrementalProcess}
\]

Since not at all points in time a process must be accessing the model knowledge, the
function is partial. The schema Access finally specifies that only one process of the
model can actually access the available knowledge of the model.

\[
\begin{align*}
\text{Access} & \ \\
\text{View} & \ \\
\text{q} &: \ IncrementalProcess \\
\langle \text{proc} \rangle \in \text{model.cascade} & \\
\langle \text{q} \rangle \in \text{model.cascade} & \\
\forall t : \mathcal{T} \ ullet \ & \text{proc} = \text{access}(t) \land q = \text{access}(t) \Rightarrow \text{proc} = q
\end{align*}
\]

This restriction can be relaxed if two processes \( p_i \) and \( p_j \) access disjoint parts of the
model knowledge, ie \( \text{view}(p_i, \text{mk}) \cap \text{view}(p_j, \text{mk}) = \emptyset \), because then there is no
danger of racing problems, or if they do not modify the model knowledge.
6.2 Referential nets

Since referential nets were already described in section 4.3, I now continue the formalisation with the focus on how referential nets are used by INC for conceptual representations in the domain of motion events. Therefore, I will indicate in the following which parts are general for incremental models using referential nets and which are particular for the domain of motion events. After formalising referential nets as such I will define increments and finally specify increment streams and increment buffers for the use with referential nets.

Formalisation of referential nets. The language $\mathcal{L}$ on which the formalisation in the previous section is based are now referential nets.

$$\mathcal{L} == \text{ReferentialNet}$$

The notation $X == Y$ introduces an abbreviation, i.e. every occurrence of $X$ is short for $Y$.

A referential net can be referred to by a term, and it consists of refOs. The term simply is a natural number.

$$\text{ReferentialNet} == [\text{term} : \text{RefNTerm}; \text{refN} : \text{P RefO}]$$

$$\text{RefNTerm} == \text{N}$$

RefOs are defined by the following schema.

$$\begin{align*}
\text{RefO} &::= \text{RefOTerm} \\
\text{RefOTerm} &::= \text{RefOType}\times\text{N}
\end{align*}$$

The definition in Habel (1986) is a bit different. There, a refO is a term, e.g., $t_1$, and sort, attributes, designations are bound to the term by functions. I decided to use this variation, because the informational encapsulation in a schema makes modelling easier. Additionally, sorts are simply special attributes, while I define them as separate entities. Apart from the fact that this will simplify the definitions of sort lattice and sort frame I chose this possibility to emphasise the important role of sorts in the referential nets approach.

The term denoting a refO consists of a letter specifying the type of the refO and a natural number.

$$\begin{align*}
\text{RefOType} &::= r | t | v \\
\text{RefOTerm} &::= \text{RefOType}\times\text{N}
\end{align*}$$

This is a free type definition, i.e. a type is defined by enumeration of its values. The notation $\text{type} ::= x_1 | x_2 | y_1(z_1) | y_2(z_2)$ makes possible recursive type definitions by repeating $\text{type}$ in $z_1$ or $z_2$. 

\[128\]
Furthermore, $x_1, x_2 : \text{type}$, $y_1 : z_1 \rightarrow \text{type}$, and the sets $x_1, x_2, \text{ran } y_1$, and $\text{ran } y_2$ are disjoint and partition type. ($X \rightarrow Y$ is an injection from $X$ to $Y$; $\text{ran } X$ denotes the range of $X$.)

RefOs of normal type are written with the letter $r$, temporary refOs with the letter $t$. The latter are, for example, used during parsing, when the identity of an entity cannot be established immediately. If the entity is already represented by a refO in the knowledge representation the temporary refO is joined with the existing one. Finally, verbalisation refOs, which have the letter $v$, are refOs that are generated during the creation of an incremental preverbal message. These refOs store the information that is used in the verbalisation of a normal refO, ie the information from the refO that becomes part of a preverbal message. The set of verbalisation refOs is a part of the discourse memory.

Each RefO can unambiguously be identified by its unique term. Thus, the following relation holds:

$\text{refOTerm} : \text{RefOTerm} \rightarrow \rightarrow \text{RefO}$

$X \rightarrow Y$ denotes a bijection from $X$ to $Y$.

With the operation GetRefO a refO can be retrieved from all available refOs by means of its term.

$\text{GetRefO}$

<table>
<thead>
<tr>
<th>t? : RefOTerm</th>
</tr>
</thead>
<tbody>
<tr>
<td>r! : RefO</td>
</tr>
<tr>
<td>r!.term = t?</td>
</tr>
</tbody>
</table>

Since each refO belongs to exactly one referential net the refNet variable of the RefO schema stores the term by which this referential net can be identified. (This is not necessary, because each existing refO can be found by searching all existing referential nets, but it makes operations more efficient.)

Extending the original definition of referential nets, a refO has an activation value as compulsory value, which is a value from $\mathbb{Z}$ (the set of integers).

The refO sorts for the domain of motion events used in 1NC and in particular for the representations of the example in chapter are defined as follows.

Sort ::= $\top$ | $\bot$ | Situation | Event | Process | State | Object | Plane | Walkway | Gate | Spatial_entity | Path | Location | Direction

* Usually, activations are rational or real numbers between 0 and 1. The reason why I chose to use the set of integers here is simply that in $\mathbb{Z}$ only integers and natural numbers are predefined, and defining rational or real numbers would add nothing interesting for the purpose at hand.
This definition corresponds to the sort lattice given in figure 4.1. $\top$ denotes the top-sort, $\bot$ the bottom-sort. The hierarchical relations between the sorts are established via the function $\text{topSort}$ that states which sort is the top-sort of another one, for example $\text{topSort}(\text{Situation}) = \{\text{Event, Process, State}\}$ means that Situation is the next higher sort in the lattice of Event, Process, and State.

<table>
<thead>
<tr>
<th>$\text{topSort}$: Sort $\rightarrow$ P Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{topSort}(\top) = {\text{Situation, Object, Spatial_entity}}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{Situation}) = {\text{Event, Process, State}}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{Event}) = {\bot}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{Process}) = {\bot}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{State}) = {\bot}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{Object}) = {\text{Plane, Walkway, Gate}}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{Plane}) = {\bot}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{Walkway}) = {\bot}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{Gate}) = {\bot}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{Spatial_entity}) = {\text{Path, Location, Direction}}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{Path}) = {\bot}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{Location}) = {\bot}$</td>
</tr>
<tr>
<td>$\text{topSort}(\text{Direction}) = {\bot}$</td>
</tr>
</tbody>
</table>

Apart from the sort, which is not defined as attribute here, a refO can have the following attributes in the domain of motion events:

$\text{RefOAttribute} ::= \text{verb_of}(\text{RefOTerm}) \mid \text{parts}(\text{seq RefOTerm}) \mid \text{part_of}(\text{seq RefOTerm}) \mid \text{concat}(\text{seq RefOTerm}) \mid \text{status}(\text{Status}) \mid \text{pe}(\text{N}) \mid \text{punctual} \mid \text{complete} \mid \text{incomplete} \mid \text{before}(\text{RefOTerm}) \mid \text{after}(\text{RefOTerm}) \mid \text{equal}(\text{RefOTerm}) \mid \text{meets}(\text{RefOTerm}) \mid \text{met_by}(\text{RefOTerm}) \mid \text{overlaps}(\text{RefOTerm}) \mid \text{overlapped_by}(\text{RefOTerm}) \mid \text{during}(\text{RefOTerm}) \mid \text{contains}(\text{RefOTerm}) \mid \text{starts}(\text{RefOTerm}) \mid \text{started_by}(\text{RefOTerm}) \mid \text{finishes}(\text{RefOTerm}) \mid \text{finished_by}(\text{RefOTerm})$

$\text{Status} ::= \text{regular} \mid \text{expected} \mid \text{discarded}$

The attribute verb_of is used only for verbalisation refO$s$. It contains the reference to the (normal) refO of which this refO is a verbalisation, for instance verb_of($r_2$) as attribute of $v_5$ means that $v_5$ is a verbalisation of $r_2$. The parts and part_of attributes serve to establish hierarchical relations between refO$s$; the refO$s$ contained
in the sequence of parts are the refOs contained by the refO having the attribute; part_of represents the inverse relation. concat is a special version of parts for path refOs that additionally encodes the sequence in which the segments of the path are concatenated. The status attribute indicates whether a refO was actually perceived (regular), which includes matches of more complex refOs with doa = 1, cf section 10.3 whether it is expected (due to a doa < 1), or whether it was expected, but the expectation was not fulfilled, so that it was discarded. If a refO represents a perceived entity the pe attribute specifies a number by which the ppu can later refer to the same perceived entity, eg if it must be updated. The attributes punctual, complete, and incomplete† determine the temporal characteristic of a situation refO. punctual says that a situation has no temporal extension but represents only a point in time. Situations with a time interval are either complete, ie they are finished, or they are incomplete, ie they are ongoing or their endpoint has not been observed. The remaining attributes represent the temporal relations to other situation refOs specified in table 4.3 on page 69. For reasons of brevity I only give the relations between extended events; the relations between punctual and between extended and punctual events as described in section 4.5 are defined analogously.

Finally, refOs can contain designations:

Designation ::= N((Name × P DesigAttribute × Activation)) | D((Description × P DesigAttribute × Activation))

Designations are either names (N) or descriptions (D), both of which can contain designation attributes and both of which – like refOs – have an activation value. However, for the purposes at hand designation attributes are not used‡.

DesigAttribute ::= EmptyDesigAttribute

EmptyDesigAttribute is only a means to keep up the overall structure of the definition. If designation attributes are actually used in a specification, EmptyDesigAttribute must be replaced by the possible designation attributes.

The activation of a designation is – analogously to the activation of a refO – defined as an integer:

Activation == Z

In the example of chapter 1 there is only one name. (Names have to be defined for the scenes that are modelled.)

Name ::= B21

* The part_of attribute is not necessary but only serves to enhance efficiency. The refOs that a particular refO is part of could be found by searching the entire referential net.
† Due to syntactic restrictions of Z the +/− of the complete-attribute is expressed as the two attributes complete and incomplete.
‡ In NC there is one exception, which is due to a technical necessity, cf section 10.2. However, the problem could also be used by other – if more cumbersome – means.
Descriptions are of the form introduced in section 4.3.

Description == Operator \times Variable \times Predicate \times seq Argument \times Activation

Thus, a Description consists of an operator, a variable, a predicate name, the arguments of the predicate, and an activation value. Alternatively to this definition, a Description could also be defined by the following schema.

\[
\begin{align*}
\text{Description} & \quad \text{op} \quad \text{var} \quad \text{pred} \quad \text{args} \quad \text{act} \\
& \quad \operatorname{Operator} \quad \operatorname{Variable} \quad \operatorname{Predicate} \quad \text{seq Argument} \quad \operatorname{Activation}
\end{align*}
\]

Yet, since this makes possible to use variants of the schema (\text{\Delta Description}) and descriptions are not meant to change, I choose the former possibility.

There are four description operators.

\[
\text{Operator} ::= \iota \mid \eta \mid \text{some}_t \mid \text{all}_t
\]

The variables are – as always – defined as lowercase letters. (Usually, there is an infinite number of variables, cf also Habel [1986]. I define it differently here, because this is a very simple definition, and for my purposes I never need more than a few variables.)

\[
\text{Variable} ::= x \mid y \mid z
\]

The description predicates in the example are the following ones.

\[
\begin{align*}
\text{Predicate} & ::= \text{be}_\text{at} \mid \text{chpos} \mid \text{curved} \mid \text{docking} \mid \text{finalpoint} \mid \\
& \quad \text{gate} \mid \text{goal} \mid \text{next}_\text{to} \mid \text{plane} \mid \text{start} \mid \\
& \quad \text{startpoint} \mid \text{stop} \mid \text{straight} \mid \text{to} \mid \text{walkway}
\end{align*}
\]

In order to differentiate sorts and the description predicates of the same name the sort names are written with a capital letter. This problem does not arise in the original definition of referential nets, because in contrast to Z it allows identical names that belong to different types / sets. The sort frame for these predicates is defined as the function sortFrame.
According to the sort frame we can now distinguish sort-correct expressions from sort-incorrect ones, cf section 4.3. One of the mentioned inconsistencies that can be modelled by referential nets are now expressions that are sort-incorrect.

\[
\text{sortCorrect} : \text{Predicate} \leftrightarrow \text{seq Sort}
\]

\[
\forall p : \text{Predicate} \cdot \exists, s : \text{seq Sort} \cdot \text{sortCorrect}(p) = \text{sortFrame}(p) = s
\]

Finally, the arguments of a description predicate are either variables (V) or references to other refOs (R).

**Argument ::= V(V(\text{Variable}) | R(\text{RefOTerm}))**

In order to adapt this specification to a domain different from motion events the following definitions have to be changed: Sort, topSort, RefOAttribute, DesignAttribute, Name, Predicate, and sortFrame.

**Operations on referential nets.** The three operations Add, Delete, and the simple Update introduced in section 6.1 can now be specified for referential nets. First of all, these operations must be differentiated between those operating on a referential net and those operating on a refO. The operations on a referential net consist of adding and deleting a RefO and updating the ReferentialNet by updating a RefO.

\[
\text{AddRefO} \equiv [\Delta \text{ReferentialNet}; r? : \text{RefO} \mid \text{refN' = refN} \cup \{r\}]
\]

\[
\text{DeleteRefO} \equiv [\Delta \text{ReferentialNet}; r? : \text{RefO} \mid \text{refN' = refN} \setminus \{r\}]
\]

133
The schema \texttt{UpdateRefO} that is used for updating the attributes and designations of a refO in \texttt{UpdateRefN} will be introduced below. The predicate part of the schema ascertains that the \texttt{refOToUpdate} is contained in the referential net and that it is identical to the refO that is updated. For convenience I define a fourth operation, \texttt{ReplaceRefO}, that differs from \texttt{UpdateRefN} in that a refO is \textit{replaced} by another one, while an update only \textit{changes} a refO:

\[
\text{ReplaceRefO} \equiv [\Delta \text{ReferentialNet}; r\?, u\? : \text{RefO} \mid \text{refN}' = \text{refN} \setminus \{r\?\} \cup \{u\?\}]
\]

When a refO is changed we must not only distinguish between adding, deleting, and updating but also whether an attribute or a designation is affected by the operation:

\[
\begin{align*}
\text{AddAttribute} & \equiv [\Delta \text{RefO}; a\?: \text{RefOAttribute} \mid \text{attributes}' = \text{attributes} \cup \{a\?\}] \\
\text{DeleteAttribute} & \equiv [\Delta \text{RefO}; a\?: \text{RefOAttribute} \mid \text{attributes}' = \text{attributes} \setminus \{a\?\}] \\
\text{UpdateAttribute}_\text{Simple} & \equiv [\Delta \text{RefO}; a\?, u\?: \text{RefOAttribute} \mid \text{attributes}' = \text{attributes} \setminus \{a\?\} \cup \{u\?\}] \\
\text{AddDesignation} & \equiv [\Delta \text{RefO}; d\?: \text{Designation} \mid \text{designations}' = \text{designations} \cup \{d\?\}] \\
\text{DeleteDesignation} & \equiv [\Delta \text{RefO}; d\?: \text{Designation} \mid \text{designations}' = \text{designations} \setminus \{d\?\}] \\
\text{UpdateDesignation}_\text{Simple} & \equiv [\Delta \text{RefO}; d\?, u\?: \text{Designation} \mid \text{designations}' = \text{designations} \setminus \{d\?\} \cup \{u\?\}]
\end{align*}
\]

More specific operations are feasible that change only the arguments of an attribute or a description. For example, the \texttt{parts} attribute has refO list as argument. There should be operations that allow not only to replace the attribute as a whole but, for instance, append a further refO to the list. Such specialised operations are used in the implementation of \texttt{rnC} but will not be specified here, because this would be dragging out the specification by adding tedious details.

\textit{Increments.} An increment was up to now defined as a piece of information that is the input and/or output of an incremental process, cf term \texttt{\textit{\ldots} on page 90} and as word of the formal language \(\mathcal{L}\) over which the incremental model is defined.
Since refOs are the main means to structure a referential net, a proper increment is a refO, and an update increment is the modification (update) of a refO. Apart from UpdateRefO, the following definitions are preceded by an R, because they were already given above and in order to indicate that these are instantiations of the schemas for referential nets. This is a standard method in Z for refining a specification.

An increment is either a proper increment (an increment that is no update increment) or an update increment. Proper increments are simply refOs.

\[
\text{R\_increment ::= Proper\{R\_ProperIncrement\} | Update\{R\_UpdateIncrement\}}
\]

\[
\text{R\_ProperIncrement ::= RefO}
\]

An update increment consists of a refOTerm that stands for the term that identifies the refO to be updated (remember that each refO has a unique refOTerm) and four sets that contain the attributes and designations that are added to or deleted from the refO.

\[
\begin{align*}
\text{R\_UpdateIncrement} & \text{-------------------------------------------} \\
\text{refOTerm : RefOTerm} & \\
\text{addAttributes : P AddAttribute} & \\
\text{deleteAttributes : P DeleteAttribute} & \\
\text{updateAttributes : P UpdateAttribute\_Simple} & \\
\text{addDesignations : P AddDesignation} & \\
\text{deleteDesignations : P DeleteDesignation} & \\
\text{updateDesignations : P UpdateDesignation\_Simple} &
\end{align*}
\]

The update operation itself takes a refO and an update increment. The predicate ascertains that the refO that is updated is actually the one the update increment was generated for.

\[
\begin{align*}
\text{UpdateRefO} & \text{-------------------------------------------} \\
\text{R : \Delta RefO} & \\
\text{u? : R\_UpdateIncrement} & \\
\text{R\_term = u?.refOTerm} &
\end{align*}
\]

More elaborate and efficient operation schemas can be defined, eg for adding a refOTerm to a parts attribute. In the implementation of iNC such operations are used; here, though, I will not introduce them in order to keep the formal specification as concise as possible.

**Operations on increment streams and buffers.** Specifying increment streams and increment buffers for referential nets is now straightforward. Increment streams are sequences of R\_Increments.

\[
\text{R\_IncrementStream ::= [stream : seq R\_Increment]}
\]
The operations \texttt{R Append} and \texttt{R Fetch} are defined analogously to the ones in section 6.1.

\begin{align*}
\texttt{R Append} & : \sim \texttt{R IncrementStream} \\
& \quad \quad \quad \sim \texttt{R Increment} \\
& \quad \quad \quad \texttt{r?} : \texttt{R Increment} \\
& \quad \quad \quad \texttt{stream'} = \texttt{stream} \sim \langle \texttt{r?} \rangle
\end{align*}

\begin{align*}
\texttt{R Fetch} & : \sim \texttt{R IncrementStream} \\
& \quad \quad \quad \sim \texttt{R Increment} \\
& \quad \quad \quad \texttt{r!} : \texttt{R Increment} \\
& \quad \quad \quad \texttt{stream'} = \texttt{tail stream} \\
& \quad \quad \quad \texttt{r!} = \texttt{head stream}
\end{align*}

An increment buffer is a special increment stream of limited size.

\begin{align*}
\texttt{R IncrementBuffer} & : \sim \texttt{R IncrementStream [buffer/stream]} \\
& \quad \quad \quad \texttt{maxLength} : \texttt{N} \\
& \quad \quad \quad \# \texttt{buffer} \leq \texttt{maxLength}
\end{align*}

Remember that more than just the two operations above can be defined on increment buffers. In particular the update operation (\texttt{UpdateRefO}) can be applied to increments in buffers. For \texttt{rsC} there are three further operations on increment buffers: append, reorder, and replace, which are needed for manipulating the traverse buffer. The append operation is the same as the one above except that it must be ascertained that the maximal buffer length is not exceeded. If the buffer already is of maximal length before the append operation the first element is dropped.

\begin{align*}
\texttt{R AppendToBuffer} & : \sim \texttt{R Append [buffer/stream, buffer’/stream’]} \\
& \quad \quad \quad \sim \texttt{R IncrementBuffer} \\
& \quad \quad \quad \# \texttt{buffer} \geq \texttt{maxLength} \Rightarrow \texttt{buffer'} = \texttt{[tail buffer]} \sim \langle \texttt{r?} \rangle \\
& \quad \quad \quad \# \texttt{buffer} < \texttt{maxLength} \Rightarrow \texttt{buffer'} = \texttt{buffer} \sim \langle \texttt{r?} \rangle
\end{align*}

The reordering operation swaps the positions of two increments in the increment sequence:
The replace operation, finally, exchanges one increment by another increment:

\[ R_{\text{Replace}} \]
\[ \Delta R_{\text{IncrementBuffer}} \]
\[ \text{pref, suff : seq } R_{\text{Increment}} \]
\[ \text{old?, new? : } R_{\text{Increment}} \]
\[ \text{buffer = pref } \lhd (\text{old?}) \rhd \text{middle } \lhd (\text{r}_1?) \rhd \text{suff} \]
\[ \text{buffer'} = \text{pref } \lhd (\text{new?}) \rhd \text{middle } \lhd (\text{r}_2?) \rhd \text{suff} \]

This completes the formalisation so that now we can start building a model of an incremental conceptualiser. In other words, now I could start describing the internal workings of \( \text{incC} \). However, before I turn to that, I first want to elaborate on the issue that is the main motivation for using incrementality in the first place, the notion of resources.
Resources are one of the main motivations for using incremental processing – apart from the simultaneous processing of input and production of output. Yet, I never explicitly stated what exactly is meant by this term and how it contributes to the cognitive adequacy of my approach. The most important point is that the behaviour of a cognitively adequate model is influenced by the amount of available resources. Since resources are limited, no perfect or unbounded rationality is possible. Instead, bounded rationality must be assumed.

I will first introduce some important terms for discussing resources in section 7.1. In section 7.2 I describe bounded rationality and argue why this kind of rationality must be regarded as cognitively adequate in contrast to unbounded rationality. Another well-known approach for saving resources are anytime algorithms, and, as I already pointed out in chapter 5 there are similarities between incremental and anytime processing. However, there are also significant differences as I will show in section 7.3. The problem of limited resources in a dynamic system is not only a relevant problem in a cognitive context but a general problem of computability. A standard A1 problem in this respect is the frame problem, especially in the form of the persistence problem, which I will discuss in section 7.4.

Before I describe INC in part C, one issue remains, viz why I designed INC ‘from scratch’ instead of using an already existing unified cognitive architecture. This is the topic of the second part of this chapter. In section 7.5 I will provide an overview of some major unified cognitive architectures and describe their commonalities. In section 7.6 I will point out how such architectures differ from the approach taken in INC and argue why, for the purposes at hand, unified cognitive architectures are too restrictive. I will also take up the issue of modularity again. Finally, in section 7.7 I will conclude the discussion of the first two parts by describing the architectural principles underlying INC.

7.1 Terminology

My terminology for resources is based on Jameson (1997) and Jameson & Buchholz (1995). In this view, resources are a very wide notion.
Term 7.1 Resource. A resource is a – material or immaterial – auxiliary means required by a task in order to perform a function in achieving a goal. The performance or even the overall success of the task depends on allocation of the required resources, or, how much of the required resources are allocated to the task.

Resources cannot be defined in general but only for a specific task and the goal that the task has to achieve. Typical resources are physical objects, human abilities, information, energy, and time (Jameson & Buchholz 1998: 96). Some resources are consumed when they are used, eg energy or time, while others are not, eg information. Finally, some resources can be divided, eg time, while others can only be used as a whole, eg a telephone number.

For incremental processing the limitations of two resources are especially of interest: time and storage capacity. In particular increment buffers are suited to model these limitations: storage capacity by restricting the number of items that can be stored in the buffer, time by the length of time an increment remains in the buffer until it is removed for further processing.

In order to be used an resource must be allocated to a task.

Term 7.2 Allocation. The allocation of a resource is the assignment of (part of) the resource to a task in order for the task to be carried out.

With respect to allocation the notion of resources is too general for the following purposes, because information is not exhausted when it is used. Hence, it would be misleading to call this allocation (Jameson & Buchholz 1998: 97). For the present purpose I therefore narrow down this term to resources that are exhausted when they are used. This excludes in particular knowledge in the following.

Allocation of resources is usually carried out with the goal to maximise (optimise) their usefulness, ie the available resources should be used in a way that maximal gain is achieved with them. The means to measure this usefulness are metrics.

Term 7.3 Metric. A metric determines the degree of usefulness of a resource for performing a task.

The use of metrics is one important difference between incremental and anytime processing. While anytime algorithms typically perform explicit estimations of the optimal use of the available resources, incremental algorithms typically only change their behaviour if a resource is in danger of becoming too scarce. For example, if working memory or a specialised buffer are close to being completely filled the system may decide to switch to another level of granularity in order to cope with the available resources. This is called adaptation.

Both of these limitations play an important role in rNC, see section 8.4 on the parameters that determine the length of the traverse buffer and the time that it stores an increment. In section 9.2 the role of the traverse buffer for the functioning of rNC is described in detail.
Term 7.4 Adaptation. Adaptation is the adjustment of the behaviour of a computing device in a way that its available (allocated) resources suffice to accomplish a task.

Agents computing resource allocation are usually divided into two main components, cf, for example, the agents by Larson & Sandholm (2001). The first component does the computation proper, ie the computation the agent was designed for. This is often done with anytime algorithms, cf section 7.3 The other component computes how much of the available resources the agent spends on such a computation. In contrast, incremental models as they were defined in chapter 5 do not have the ability to reason about resources, and, consequently, are no subdivided into such components. Therefore, adaptation will play only a minor role in the following. In particular, tNC is not capable of regulating the allocation of its resources itself, and the amount of available resources are fixed when the system is started and do not change afterwards. A possible extension is to monitor the fill-rate of the traverse buffer, see section 9.2. If it is permanently filled so that elements get lost tNC may change its strategy in that the elements remain in the buffer for a shorter time span.

7.2 Bounded rationality

For cognitive models it is particularly important to cope with the limitations of the available resources. A particularly important idea was put forward by Herbert Simon: when an agent makes decisions, it does not behave globally optimal, ie he/she/it does not make the optimal decision with respect to the current state of the world (Simon 1955, 1982). This is due to the limited amount of resources it has available, most of all deliberation time and knowledge about the world. Thus, it has to make the best of the resources it has available. As a consequence, its decision is not perfectly rational but the rationality is bounded (Simon 1982: 409, vol 1).

So, Simon considered unbounded rationality as unrealistic due to the limitations of the natural agent, especially the ones of computational and predictive ability, which are caused by the agent’s limited knowledge and by the complexity of the environment (Simon 1982: 241, vol 2). Hence, models based on unbounded rationality produce unrealistic and, therefore, unreliable predictions of human behaviour. In order to adequately investigate and model people’s behaviour as it occurs in real-

* I will use it in the following, because, firstly, it solves the unsolvable he or she or he or she or he and she or even s/he problem and, secondly, because it emphasises the fact that my goal is to build a model of an artificial agent. However, it should be understood as often encompassing natural (he or she) agents without explicitly mentioning it.

† Simon’s idea is mainly directed against the notion of a perfect, unbounded rationality, which was the prevalent theory at that time he first proposed the idea, ie in the 1940’s and 1950’s. And looking at most of the current theories on decision making one gets the impression that despite the fact that Simon’s theories were and are widely acknowledged this situation did not really change. For example, the highly praised Art book by Russell and Norvig still dismisses the notion of bounded rationality, because they want to built agents that are ‘actually useful in the real world, whereas […] satisficing agents might or might not be, depending on their own whims.’ (Russell & Norvig 2003: 973)
ity Simon proposed not to make ideal world assumptions but rather to study the exhibited human behaviour directly in order to obtain realistic results.

Based on this analysis Simon considered humans as *satisficing* rather than optimising beings. Satisficing means that humans often do not consider all available possibilities; rather they consider them one at a time and stop the search 'once they discover the one that they regard as satisfactory' (Simon 1982: 413, vol 2). However, there is another kind of bounded rationality that fits even better to incremental processing than satisficing, namely the *fast and frugal heuristics* by Gigerenzer, Todd, & the ABC Research Group (1999), Todd & Gigerenzer (2000). In this approach making a decision consists of three major components:

1. heuristic principles guiding search,
2. heuristic principles for stopping search,
3. heuristic principles for decision making.

Although Gigerenzer et al have classical decision making problems in mind, eg deciding which of two cities is bigger (without knowing the number of inhabitants), the overall approach is similar to incremental processing as I laid it out in section 5.3. There are two heuristic principles guiding search: firstly, the overall principle that each input increment is taken and processed in relation to the available knowledge, and, secondly, the heuristics that establish the local contexts on which the incremental algorithms operate. The heuristic principles for stopping search correspondingly are that a new input increment is available or that the results of the incremental algorithm are not improved any more. The heuristic principles for decision making are integrated in the way the incremental algorithms work, for example, the selection strategy that decides which event is verbalised.

I do not want to go into further detail here, but it should be noted that the fast and frugal heuristics by Gigerenzer et al do not lead to a decreasing quality of the results. On the contrary, the performance of their methods is comparable to that of techniques using unbounded rationality, eg multiple linear regression. Yet, in addition to that their approach is much more robust with respect to new input data, a point very much in favour with regard to cognitive adequacy, because while it can be tolerated that the best decision possible is not found, it must be able to make decisions of constant quality in different settings and environments.

Hence, incrementality is one way of enabling humans to act despite their limited resources in a bounded rational way, which is not only cognitively adequate but also robust.

### 7.3 Anytime processing

The use of bounded rationality can greatly enhance the cognitive adequacy of a model, because an agent can behave in a rational way without the unrealistic assumptions of unlimited resources. As I already mentioned, *anytime* algorithms were
developed for very similar reasons. In particular they enable agents acting in an environment of a realistic (natural) complexity to reach decisions in an acceptable time. Thus, they reduce the required time as well.

The term *anytime algorithm* was coined by Dean and Boddy [Dean & Boddy 1988, Boddy & Dean 1994]. Anytime algorithms are algorithms that improve the quality of their results over time, i.e., the more time they have available the better the result is. They are used, for instance, for constraint satisfaction problems, the interpretation of sensory data, and path planning like in the travelling salesman problem.

The defining property of an anytime algorithm is that it can be stopped at any time to provide a solution, and the quality of the solution increases with computation time. This property allows a trade-off between computation time and solution quality, making it possible to compute approximate solutions to complex problems under time constraints. (Hansen & Zilberstein 2001: 14)

According to this quotation, anytime algorithms extend the usual notion of computation by being able to return not only one result but a range of results, the quality of which can be determined by different metrics. Three metrics have been proving particularly useful [Zilberstein 1996: 74]: firstly, *certainty* is the metric that measures the degree of certainty that a result is correct; secondly, *accuracy* measures how close the result is to the global optimum; thirdly, *specificity* determines the level of detail of the result. The last metric presupposes that the algorithm only produces correct results, only the level of detail increases for each result.

The point in time when an anytime algorithm stops can either be determined beforehand, e.g., by stochastic evaluations of previous executions, or by monitoring its progress [Hansen & Zilberstein 2001]. Accordingly, there are two kinds of anytime algorithms, *interruptible* and *contract* algorithms. The former can be interrupted at any time, e.g., by a signal from the outside, and present the (best) result they computed so far; the latter know the amount of time in advance, thus, the optimisation of result quality and run-time is performed beforehand. Anytime processing raises a new problem not present in incremental processing, the *meta-level control problem*: ‘the problem of determining the stopping time for an anytime algorithm that optimizes the expected value of computation.’ (Hansen & Zilberstein 2001: 14)

According to [Zilberstein 1996: 74] an anytime algorithm should have the following properties:

1. *measurable quality*: the quality of the result can be determined precisely,
2. *recognisable quality*: the quality of the result can be determined at run-time,
3. *monotonicity*: the quality of the result increases with time and input quality,

If quality is recognisable the anytime algorithm can guarantee monotonicity by returning the best result generated so far rather than the last result generated.
4. **consistency**: the quality of the result is correlated with time and input quality,
5. **diminishing returns**: the improvement in quality is larger at the early stages of the computation and diminishes over time,
6. **interruptibility**: the algorithm can be stopped at any time and provide an answer,
7. **preemptability**: the algorithm can be suspended and resumed with minimal overhead.

As this list shows, the main concern of anytime processing is the quality of the result. Most of the properties are desirable for incremental algorithms as well, which is one of the reasons why incrementality and anytime are kindred processing mechanisms. In particular both are similar in the following respects:

- they need less resources for producing their results than algorithms that search for the global optimum,*
- they can produce output before the computation has yielded the optimal result,
- they do not ascertain to produce the globally optimal result, ie they are methods of bounded rationality.

Yet, incremental and anytime algorithms also differ in important respects:

- incrementality does not know the meta-level control problem,
- anytime ‘is about’ the quality of a result, incrementality about producing output at a constant rate (which can be identical),
- anytime processes are used for tasks in which the result can be improved gradually; incremental processes produce output of (more or less) constant quality – although corrections are possible,
- in anytime the resources that are spent on a computation are the object of computation itself, ie the model reasons about how what amount of time it should allocate for performing a task (thus, there are really two kinds of computations); in incrementality resources are ‘just used’ (but adaptations may be performed),
- anytime processes usually work *on demand*, ie they are called in order to return a result; incremental processes are more autonomous in that they are not called from ‘the outside’ but work at a steady rate and produce output as soon as it is available†,
- anytime processes return the best result they have got up to that point, while incremental processes just produce output as quickly as possible, at least with respect to Extended Wundt’s Principle,‡
- anytime algorithms usually do not deal with dynamic input in a way where the

* However, there are cases in which incremental processing needs more resources than a corresponding non-incremental mode of processing, cf the remark on priming in section §5.3
† This is true at least for Extended Wundt’s Principle but may be seen differently in approaches that perform separate *when-to-say* computations, cf Kilger & Finkler (1999).
‡ If Extended Wundt’s Principle is not used and if in particular incremental processes generate some candidate results before deciding upon one of them, anytime and incremental processing operate identical in this respect.
difference of new and present knowledge is considered; this is what incremental processing is all about.

However, although there are substantial differences between anytime and incremental algorithms both can be combined in order to build more reliable systems. For example, [Menzel 1994] describes a constraint-based approach to natural language parsing that uses anytime techniques in combination with incremental parsing. In particular, he describes how weak anytime algorithms can be utilised for this purpose. Weak anytime algorithms are anytime algorithms for which it is not possible to establish the optimal trade-off between output quality and required run-time before the algorithm starts computing. The progress of these algorithms is measured by determining the remaining ambiguity of the parsed language. The goal is to generate a representation without ambiguities. This parsing technique proves to be more robust and fault-tolerant in comparison to incremental systems that use no anytime mechanisms. For instance, parsing of ungrammatical language is performed by these algorithms without extra effort.

7.4 A remark on the frame problem

The frame problem was first observed by [McCarthy & Hayes 1969]. Originally it termed a quite narrow problem in the situation calculus. The situation calculus is a first order logic for reasoning about time and a changing world, see [McCarthy & Hayes 1969] for the original description and [Reiter 2001] for a recent version. Within this calculus actions define changes of states of the world, e.g. in a model of the well-known blocks-world it is possible to describe how the representation of the world changes if a block is moved on top of another one. Since this action does not change size or colour of the blocks, such non-changes must be specified by so-called frame axioms, e.g. the size of a block is not changed by movements. Each time an action is carried out the changes as well as the non-changes must be computed. Since there normally are many more facts that do not change than facts that do change, there is a large number of frame axioms. This leads to an explosion of complexity, which requires to address the frame problem in the discussion of resources.

The original frame problem has lead to lots of discussions about this particular problem of the situation calculus and about related problems in this and other calculi, cf. [Morgenstern 1996] for a concise overview and [Shanahan 1997] for a history of proposed solutions to the frame problem including a further proposal. The reason is that all calculi dealing with changeable representations eventually get this or a similar problem. I do not want to discuss the frame problem as such here, but there is one point in incremental processing that is subject to it, namely the point where a heuristic determines the local context for an incremental algorithm. The

*Clark 2002* discusses a similar version of the frame problem and proposes to use an intriguing technique by [Kleinberg 1998] to solve it. The upshot is that in a densely interlinked knowledge
assumption up to now is that all relevant information that is required for the incremental algorithm to make its computation is available in the local context, which leads to a reduction of the required resources. Thus, this method could already be considered a solution of the frame problem, because only few frame axioms have to be applied. This is even more true, because it is supported by the most obvious solution to the frame problem, which was already suggested by McCarthy & Hayes (1969). This solution consists in proposing the general axiom that no facts are changed by an action, except those that are explicitly modified.

Morgenstern (1996) argues that although this is the original frame problem it is not the real one, and the solution suggested by McCarthy and Hayes is no real solution. Instead the real frame problem is the persistence problem, the problem to decide which facts are changed by an action and which persist. To my knowledge no general solution of the persistence problem has been found yet. The solution I will use here is, therefore, not to solve it at all. Instead, the goal for a cognitively adequate model must be to build a model that is ‘good enough for government work’ as Dennett (1996: 6) puts it.

My goal, therefore, is to build a model that works ‘according to the ubiquitous biological design principle: oversimplify and self-monitor.’ (Dennett 1996: 4) This design principle can be read in two ways. Firstly, a model design need not be perfect right from the start. A first version is built and then the cases in which it does not behave like it should are examined and the next, better version is built. † In nature this is done by the learning of an organism and by evolution in a population. The second reading of this principle is that a model needs mechanisms that enable it to correct errors, because it should be robust. This is done by the monitor proposed by Levelt (1989), cf also chapter 15.

### 7.5 Unified cognitive architectures

Apart from cognitive models that are built to perform one specific task there is already a long tradition of unified theories of cognition as they were called by Newell (1990). I will refer to the means by which such a unified theory is simulated as unified cognitive architecture. The first of these architectures was the General Problem Solver (GPS) developed by Newell & Simon (1963), which can still be regarded as the main inspiration for current architectures. Since the field of cognitive modelling owes a lot to these approaches, I will sketch their main tenets in this section.

representation like the world wide web much can be gained by not evaluating the information but the structure of the information. Put informally, in order to find an entity one has to look at the links not the linked entities. A similar solution can be used for referential nets.

* From a cognitive standpoint the frame problem seems to be an ‘artificial artificial intelligence’ problem. The problem is artificial (ill-posed), because to solve it means to build agents that perform better than those found in nature. Given that natural organisms are particularly adapted to their environment it remains to be seen whether that is possible.

† In fact, most software and system development is done this way.
In the next section I will then show the differences to the approach taken here and argue why for the purpose at hand a unified cognitive architecture would only be of limited use.

Models like \textsc{rnC} are subject to a major criticism initially brought forward by Newell and being a main inspiration for proposing a unified cognitive architecture in the first place. The criticism is that such models always only capture one aspect of cognition, and the claim is that it is no (hard) problem to build a model that fits a set of data. More specifically there are three main issues:

(a) the problem of \textit{irrelevant specification} (in a complex computer program, which of the myriad aspects of the program carry theoretical content, and which are irrelevant implementation details?) \cite{Lewis_2001}; (b) the problem of \textit{too many degrees of freedom} (an unconstrained computer program can be modified to fit any data pattern); and (c) the problem of \textit{identifiability} (any sufficiently general proposal for processing schemes or representations can mimic the input/output characteristics of any other general processing or representation scheme \cite{Lewis_2001})

To cut a long discussion short: it is true, \textsc{rnC} cannot answer to these criticisms – or, at least only very weakly. Yet, before I argue why I do not use a unified architecture, which are less affected by this critique, let me first describe the main properties of these architectures, cf also \cite{Taatgen_1999}. The main consequence drawn from the criticisms above is that all unified cognitive architectures restrict the possible computations in a way they deem plausible and supported by empirical data to simulate characteristics of human cognition (or cognition in general).

\textbf{Soar.} Soar was developed by Allen Newell and is the direct successor of the General Problem Solver \cite{Newell_1990}. The most important ingredients are \textit{productions} – also called production rules or chunks –, \textit{precedence rules}, and a \textit{working memory}. Productions are condition–action pairs: each time the condition of a production is true the action is executed. This is referred to as firing of a production. Since multiple conditions can be true at a given point of time the set of \textit{precedence rules} determines the order in which these productions fire. The productions are stored in the long-term memory – also called production memory – and operate on a working memory, which contains the facts known about the world. Productions are suited well for modelling human memory, because they form a recognition memory \cite{Lewis_2001}. This means, firstly, they are associational (they are accessed by content, not an address like most computer memories), secondly, they are dynamic (they can change over time), and thirdly, they are cognitively impenetrable, because their content cannot be accessed freely.

Soar executes an infinite \textit{recognise–decide–act cycle}. This means that in the recognise step the productions are identified whose condition is true. In the decide step one of these productions is selected, which is also called the selection of the next goal. In the act step, finally, the selected production is executed. If no decision
can be made an *impasse* occurs and a new sub-goal is created to resolve it. If the system finds a solution by this sub-goaling mechanism it generates a new production which it adds to the set of productions. This learning mechanism is called *chunking*. What is learned stays available in the future; there is no ‘forgetting mechanism’.

**ACT-R.** ACT-R is an architecture similar to Soar. It was developed by John Anderson and has been published in different versions. The most recent and most complete of these is described in Anderson & Lebiere (1998). Like Soar it is first of all based on the idea of productions (chunks). Thus, ACT-R models are symbolic as well. However, ACT-R can also be seen as a hybrid theory, because it additionally models (sub-symbolic) activation: each chunk has an activation that increases with a successful use and decays otherwise.

Apart from this there are two main differences to Soar. Firstly, in ACT-R the chunk activations are responsible for determining which chunks are retrieved and how long it takes to retrieve them. Thus, there are no preference rules. Secondly, ACT-R does not distinguish the memory for productions from the working memory but uses only one memory.

**EPIC.** While Soar and ACT-R are models that are especially good as models of central cognition, the EPIC (Executive-Process / Interactive Control) architecture (Meyer & Kieras 1997a,b) stresses the peripheral areas of cognition and how they influence the performance of a system or human. This means in particular that is has several separate perceptual and motor processors, eg a vocal motor processor. Additionally, EPIC focuses on aspects of multiple-task performance, ie on issues of the parallel execution of tasks. Apart from this and the fact that it has no model of learning, EPIC works along the same lines as Soar and ACT-R, ie it uses productions and a division of long-term memory and working memory.

**PSI.** Another broadening of the goal pursued by unified cognitive architectures is to include emotions and social factors in the theory. One very elaborate model containing these additional aspects, especially an emotional component, is PSI (Dörner 1999). The approach is similar to Braitenberg (1984) in that it is an approach in synthetic or theoretical psychology: a complete agent is built up from simple devices that are combined to form more and more complex devices exhibiting more and more complex behavioural patterns.† The main difference to the other three architectures is that PSI is based on a neuronal theory of cognition, not symbols.

* Actually, long-term memory and production memory are separate memory modules in EPIC, so that together with the working memory there are three memories *in toto*.
† The aim of the PSI theory is to build an *artificial soul* that has more dimensions of adaptivity than a solely cognitive architecture. For example, emotions are understood as modulations of cognitive processes, ie they are not additional processes but processes in another dimension. For instance, the emotional state of an agent influences the degree of detail in which computations are carried out. From time to time an agent will be ‘angry’, eg caused by environmental influences, which facilitates deciding and acting quickly rather than weighing all possibilities in order to find a better action.
What these architectures have in common is:

1. the use of productions,
2. a unified working memory instead of multiple memories in different processes,
3. a division of working memory and long-term memory (except ACT-R and PSI),
4. a learning mechanism (except EPIC).

7.6 Differences to INC, or reasons for not using a unified cognitive architecture

So, if the unified architectures of cognition have all these advantages and claim to simulate important aspects of cognition that unconstrained models do not, why then not develop INC within one of these frameworks? These are really two questions:

1. Is it possible to implement INC in a unified cognitive architecture?
2. What are the advantages of not doing this?

Is it possible to implement INC in a unified cognitive architecture? Put differently: are the assumptions of INC compliant with the assumptions of unified architectures? Yes and no. On the one hand there are strong similarities between both approaches, and I consider it quite likely that INC can be implemented in a unified cognitive architecture. There are two major similarities. Firstly, INC also has a division of working memory and long-term memory: working memory for the conceptual representation of the current state of affairs and other temporary stored knowledge, long-term memory for the rules (productions) on how to construct complex concepts from simple ones.† Just like in the unified architectures the long-term memory consists of productions, while the working memory consists of a declarative memory. Secondly, the recognise–decide–act cycle of unified cognitive architectures has a counterpart in the incremental processes: identify the focussed element, determine the local context, execute the incremental algorithm.

On the other hand there are also significant differences. The answer to the second question is that INC has a somewhat different purpose than unified cognitive architectures: apart from modelling conceptualisation for language production, the processing mechanisms by which this is done, ie incremental processing, are equally major issues in its development. In unified cognitive architectures the processing mechanisms themselves are (usually) not the topic of research. I want to mention four major differences here.

Firstly, incrementality in a cascaded architecture includes a kind of parallelism different from the one in the mentioned architectures, because not ‘arbitrary’ pro-

---

* PSI uses a notion of productions that is a bit different, but for the present purpose this suffices.
† In INC these two memories are the current conceptual representation (CCR) and the concept storage (CS), respectively. Both are represented by referential nets.
ductions are considered in parallel but defined processes simultaneously work on one information stream on different stages.

Secondly, the Soar control structure, to take just one example, uses a least-commitment strategy \cite{Lewis2001}. Changing this strategy in order to comply to Extended Wundt’s Principle – which, basically, is an early-commitment strategy – would substantially change the way Soar works, ie it would change the architecture, not only the model. However, if the architecture is changed, there is no guarantee that the resulting architecture still obeys the original constraints. Similar arguments can be given for the other architectures.

Thirdly, although ACT-R, for example, contains distinct modules, they are of a substantially different nature than inC’s quasi-modular processes, cf chapter 2. Modules in unified cognitive architectures are not about ‘generation of preverbal messages’ or ‘construction of a conceptual representation’. Instead, they contain the ‘goals of the system’ (goal buffer) or the ‘set of intentions’ (intentional module). Put differently, the processes (modules) of unified cognitive architectures are general processes of cognition, not processes specialised on a particular task. (EPIC can be seen as an exception, because its perceptual processors function comparably to inC’s processes.) Yet, the cognitive processor (the central processor) in which the conceptualiser would have to be located in these architectures does not contain such specialised components.

Thus, a module (a process) in inC executes a fixed task, while in unified cognitive architectures modules are, for example, the part of the system that contains the set of goals. The model proper is non-modular. A Soar model that comes close to being modular in the inC sense is NL-Soar by Lewis \cite{Lewis1996}. It is a Soar model for natural language processing. Regarding the questions of modularity Lewis states that with respect to the processing mechanisms the model is non-modular, ie the mechanisms that are used for cognition in general are also used for language comprehension. Thus, NL-Soar is a horizontal architecture in the sense of Fodor \cite{Fodor1983}, ie an architecture that shares mechanisms and resources across domains. Yet, the model also exhibits modular properties: informational encapsulation, domain specificity, mandatoriness, and speed (operators need on the order of 100 ms), cf section 5.4. It must be doubted, however, whether the claimed informational encapsulation really stands the test, because in principle each rule can be taken to be combined with another one, or a rule can ‘evolve’ to fulfil a different function. The means for this is the learning mechanism.

With this we are back at the discussion of modularity, cf chapter 2. The question is: what does a process like selection or linearisation in inC model; a general selection or linearisation faculty or a specialised one? In the extreme case such a specialisation would be a selection for the conceptualisation of motion events for language production. In terms of Fodor \cite{Fodor1983} 10–23: are the processes horizontal or vertical faculties? The overall structure of unified cognitive architectures is further support for the view that there are modular aspects as well as non-modular ones, and the closer a component is to the periphery the more modular it is, while the components close to the central system are more or less non-modular. Thus, now
approaching this issue from the direction of cognitive modelling I come to the same conclusion: quasi-modules, ie components that are cognitively impenetrable but not informationally encapsulated (page 35), are an adequate notion of the kinds of components that have to be modelled in case of conceptualisation.

The fourth difference of unified cognitive architectures to inC is that in inC the degree of informational encapsulation of a component is an important aspect of modelling. In particular, an incremental process does not have access to all knowledge of the model. Furthermore, allowing indirect feedback has the consequence that informational encapsulation is not strict. This does not mean that in a unified cognitive architecture all knowledge can be used to solve a problem. However, as Lewis (2001) states, there are no architectural barriers to the knowledge that can be used for a computation. The consequence is that unified cognitive architectures rely heavily on the self-organisation of the (given and learned) knowledge, while in models like inC the builder of the model defines the informational encapsulation of the components during the model’s development.

The issues of modularity and informational encapsulation play an important role not only for the architecture that is used for a cognitive model but also for the issue of resources as can be seen in the following passage from Fodor (1983):

\[\text{[I]t may well be that processes of input analysis are fast, because they are mandatory. Because these processes are automatic, you save computation (hence time) that would otherwise have to be devoted to deciding whether, and how, they ought to be performed. […] W]hat you save by indulging in this sort of stupidity is not having to make up your mind, and making up your mind takes time.}\ (Fodor 1983: 64)

Summing up, although there are drawbacks for the method used here (see the criticism at the beginning of section 7.5) and although it is likely that at least a similarly working model of a conceptualiser can be constructed within a unified cognitive architecture, investigating incremental processing in the way it is done here can hardly be achieved within a unified cognitive architecture.

### 7.7 Principles used in building inC

We are now at the end of the general discussion of a suitable architecture for an incremental conceptualiser for language production that I pursued in the first two parts of this investigation. Before I will describe inC in detail in the next part, I want to lay out the architectural principles according to which it is designed.

The most important principle is Extended Wundt's Principle, which can also be referred to as massive incrementality or early commitment. It is the main mechanism by which the bounded rationality of inC is realised. Extended Wundt’s Principle means in particular that there is no explicit reasoning about the when-to-say, cf page 83. From this follows the use of simple, small, ‘dumb’ processes, which are
very fast. The complex behaviour of the model is a result of their interaction.

I will now shortly discuss the different kinds and dimensions of incrementality discussed in chapter $5$ and describe the variety of incrementality that is used in $\text{inC}$. As was already pointed out, neither left-to-right nor full incrementality is desirable in its pure form for a cognitive model. For this reason, $\text{inC}$ uses the strategy to evaluate local contexts within the available knowledge around the focussed element that step by step take different and/or more knowledge into account. The issue of word-based vs constituent-based incrementality is not really applicable here, but the underlying question of what the increments of the single processes are was already partly answered. A detailed account will be given in part $\text{IV}$. Since the notions of partial and qualitative vs quantitative incrementality have been discarded as not useful, let me just say that in those terms $\text{inC}$ is a quantitative, incremental model. $\text{inC}$ is a unboundedly incremental model, ie the amount of time that is required for processing input depends on the size of the input. (However, as already pointed out on page $58$ the size of $\text{inC}$’s input is rather constant.) Most of the incremental processes of $\text{inC}$ use linear incrementality most of the time. The most important case in which hierarchical incrementality is used is when $\text{pvm}$-generation decided to verbalise a description that refers to other refO’s, cf chapter $13$. Although preformatted incrementality is a very attractive possibility to generate subsequent preverbal messages that stress a contrast – and for this reason should be included in a cognitively adequate conceptualiser – it is not (yet) used by $\text{inC}$. Thus, $\text{inC}$ only uses free-format incrementality.

$\text{inC}$’s overall architecture corresponds to the one defined in sections $5.3$ and $6.4$. The means of structuring are processes, and since it uses update increments to keep track of the changes in the external state of affairs it is non-monotonic. Due to the lack of lookahead and the use of Extended Wundt’s Principle, buffering and indirect feedback are the means to reduce the number of updates and to enhance the robustness of the model – apart from an experimental monitoring component. The processes in $\text{inC}$ generate no parallel increments, so that the number of increments is always $1$. The parallelism of the incremental processes in a cascaded architecture means that the model is indeterministic, and the use of local contexts means that consistency is not ascertained. (However, the representations used in the current version of $\text{inC}$ are in fact consistent, which means that $\text{inC}$ has no mechanisms for resolving inconsistencies.) Finally, as was already pointed out more than once by now, $\text{inC}$ is a discrete, modular model.
INC – THE INCREMENTAL CONCEPTUALISER
AFTER DISCUSING CONCEPTUALISATION AND INCREMENTALITY THE GOAL OF THIS PART IS TO BRING TOGETHER THESE ISSUES IN THE MODEL INC. IT SHOULD BE KEPT IN MIND THAT INC IS MORE LIKE A FRAMEWORK OF AN INCREMENTAL CONCEPTUALISER THAT HAS BEEN ELABORATED IN SOME PARTS – AS TIME ALLOWED. THIS IS THE PRICE FOR HAVING A MODEL THAT COVERS A LARGE STRETCH OF HUMAN COGNITION. THUS, INC IS NO COMPLETE MODEL OF A CONCEPTUALISER BUT CONTAINS MANY GAPS AND HOOKS, IE PLACES IN THE MODEL WHERE FURTHER ENHANCEMENTS CAN AND MUST BE MADE IN ORDER TO CONSTRUCT A MORE COMPLETE MODEL OF THE CONCEPTUALISER.


8.1 PROCESSES

THE OVERALL ARCHITECTURE OF INC IS SHOWN IN FIGURE 8.1 ON THE FOLLOWING PAGE. IT CONSISTS OF FOUR MAIN PROCESSES THAT FORM THE CASCADE OF THIS INCREMENTAL MODEL (TERMS 5.1 AND 5.15 IN SECTION 5.3), ONE AUXILIARY PROCESS (THE CONCEPT MATCHER), AND TWO STORAGES (MEMORIES) FOR REPRESENTATIONS. THE FOUR MAIN PROCESSES THAT MAKE UP THE CASCADE ARE CONSTRUCTION, SELECTION, LINEARISATION, AND PVG-GENERATION (PREVERBAL MESSAGE GENERATION). THE DIVISION OF LABOUR BETWEEN THE TASKS EXECUTED BY THESE PROCESSES WAS ALREADY INTRODUCED IN SECTION 5.2 AND WILL BE ELABORATED FURTHER IN CHAPTERS 10 TO 13. IN SHORT, CONSTRUCTION READS THE OUTPUT OF THE PERCEPTUAL PRE-PROCESSING UNIT (PPU), CALLED PERCEIVED ENTITIES, AND BUILDS UP THE CURRENT CONCEPTUAL REPRESENTATION (CCR) FROM THEM WITH THE HELP OF THE CONCEPT MATCHER. (CONCEPT MATCHER AND CONCEPT STORAGE (CS) COULD ALSO BE SEEN AS PART OF CONSTRUCTION, CF SECTION 8.2.) EACH TIME CONSTRUCTION MODIFIES THE CCR IT INFORMS SELECTION OF THE CHANGE, WHICH DECIDES WHETHER THE CHANGED ELEMENT WILL BE VERBALISED. IF IT CHANGED THE PLANS FOR VERBALISATION IT TELLS LINEARISATION, WHICH CHECKS WHETHER THE SEQUENCE OF THE EVENTS SELECTED FOR VERBALISATION (THE SEQUENCE OF SUB-INTENTIONS,
Figure 8.1: The architecture of \textsc{inC}

cf page 37 needs to be reordered. \textsc{pvm}-generation, finally, takes one of the sub-intentions and generates an incremental preverbal message for it. There is no increment stream between linearisation and \textsc{pvm}-generation, because these processes coordinate themselves via the traverse buffer, which contains the items that were chosen for verbalisation by selection but for which \textsc{pvm}-generation did not generate an incremental preverbal message yet. The traverse buffer will be discussed in section 9.2.

There is a viable alternative to an architecture with a fixed cascade of processes: 'In situations involving search, it is often useful to spawn several processes which look for alternatives.' De Smedt (1990a: 142), cf also page 104. In such architectures several instances of a process run in parallel. De Smedt argues for the case of an incremental formulator as follows:

Clearly, a parallel formulator has an advantage compared to one which operates in a sequential mode. Newly incoming fragments which take little processing time can be uttered before older but more difficult fragments are ready. This suggests that the time when a fragment can be uttered depends not only on the moment when it has entered the formulator, but also on how much processing time the formulator spends on it. Extraposition of long and complicated phrases at the end of the sentence, which is often found in spontaneous speech, is indeed predicted by this model.

(De Smedt 1990a: 82)

Solutions with spawning processes raise new problems. Firstly, either the results of the processes must be integrated when the processes terminate, or the res-
ult of one process must be determined as the overall result. Secondly, processes spawned simultaneously will terminate at different points in time, i.e., there is a new temporal coordination problem. Generally speaking, this method can reduce runtime, which makes it an interesting possibility. However, spawning processes comes at the mentioned cost of increased overhead for managing the instances. From the cognitive standpoint this raises one more time the issue of the modularity of components: if there is only one module (processor) for a process (task, e.g., selection) then there can be only one instance. The processes in INC that may profit from a spawning mechanism are the concept matcher and PVM-generation, and to a lesser degree also construction.

In the implementation of INC there are maximally two instances of a process, and that only for a very short period. This happens when a process has finished one iteration of its loop and just waits for an acknowledgment that a change in the CCR has been completed before it starts the next iteration. For example, when construction inserted a new refO into the CCR, the CCR acknowledges the accomplishment of the modification. (More precisely: the acknowledgment is sent by the process in which the CCR is encapsulated, cf. chapter \[5\]) Construction has to wait for this acknowledgment so that it does not operate on an old state of the CCR. Nevertheless, it can already read the next input from the PPU and prepare it for processing. Therefore, waiting for an acknowledgment and the preparation the next input increment can be carried out in parallel. Apart from these cases, there is only one instance of a process in INC, and the mentioned coordination problems do not arise.

8.2 Representations

INC does not possess a detailed memory model, e.g., issues like learning and forgetting are not considered. Furthermore, INC makes no claims about access and storing of information, e.g., the memories are linearly searched without accounting for an organisation of memory, say, with regard to the domain. As I already stated in section \[7.6\] INC’s memory is subdivided according to the well-known distinction between working memory and long-term memory. The concept storage (CS) corresponds to long-term memory, the CCR to working memory. Put differently, the CS contains encyclopedic knowledge, the CCR situation and discourse knowledge. Thus, INC’s model knowledge consists of the CCR and the CS. With this, I use a more differentiated memory structure than [Levelt 1989].

The CCR could be split up in multiple representations so that each incremental process of the cascade contains its own memory. The arguments against this are

\[\text{The concept matcher could match the candidate list handed over by construction against multiple concepts in parallel. PVM-generation could chose designations for multiple refOs in parallel. The construction process could generate multiple candidate lists that could be sent to multiple instances of the concept matcher. Whether construction would operate quicker or more reliably when it has to evaluate multiple best matches is left to be seen. Please refer to the corresponding chapters on these processes for a description of the processing methods of these processes.}\]
that multiple memories cause unnecessary redundancies and that indirect feedback would only be possible via increment buffers. In accordance with Newell (1990: 164) I therefore assume only one integrated memory.

The ccr is not subdivided further into articulatory buffer, visuo-spatial sketchpad, and central executive (Eysenck & Keane 1995: 129ff). It can be said, however, that the latter two are relevant for conceptualisation, while the articulatory buffer need not be considered, because it is located much further down in the language production system. Hence, I consider no interferences between the articulatory buffer and other parts of working memory – if there are any. Furthermore, I will say nothing about the division of labour between the visuo-spatial sketchpad and the central executive. Nevertheless, most likely processes operating on spatial knowledge – which object is located where and which object is moving how – mainly use the visuo-spatial sketchpad and the processes that reason about what and how verbalisations are produced mainly use the central executive. Additionally, since both are located in the working memory, it is quite likely that there are interferences between the two. Nothing of this is considered in nC.

The cs can only be accessed by the concept matcher, and the concept matcher is connected only to construction. Thus, cs and concept matcher could be integrated into construction. However, the architecture as it is presented here has the advantage that the representation corresponding to long-term memory is a separate structure and that the access to it is modelled by a separate process. So, the concept matcher can be seen as interface to long-term memory.

It may be considered misleading, though, to call the distinction between the two representations in nC the one between working memory and long-term memory. Yet, nC’s subdivision is analogous to that of unified cognitive architectures, where these terms are used in this way. However, the actual distinction in nC is the one between previously available knowledge that is used for constructing a conceptual representation and knowledge constructed during by this task, the conceptual representation itself. The former knowledge is stored in the cs, the latter in the ccr.

8.3 Increments

nC uses the following kinds of increments between its processes and between processes and representations:

- perceived entities, which are the input to nC,
- the increments of incremental preverbal messages, which are the output of nC,
- information exchanged between construction and concept matcher: candidate list and best match (the latter consisting of match list and DOA, see the next section),
- notification of changes in the ccr sent from construction to selection and from selection to linearisation,
- the pointers to refOs in the ccr that are stored in the traverse buffer, including update increments for modifying the traverse buffer.
All of these kinds of increments will be described further in the following chapters when their role in the functioning of nC is explained. This list just serves to give an overview.

8.4 Parameters

nC has another important property, which is not depicted in figure [8.1]: it is a parameterised model. Values are assigned to these parameters at system start and influence nC’s behaviour. Note that the parameters only influence nC’s behaviour and do not determine it, ie despite identical values for parameters and identical input the behaviour and the generated output can differ. The reason for this is that the processes run in parallel, which results in an indeterministic behaviour of the model. Up to now, nC knows four parameters; a number of additional ones should be integrated in future versions, cf section [6.3].

Degree of Agreement Threshold (doat). The concept matcher determines the degree of agreement (doa) between a list of refOs that it obtains from construction, the candidate list, and the concepts stored in the cs. It gives back the entry with the highest doa to construction, which is the best match. Construction then decides whether the doa of best match and candidate list is high enough to insert the best match into the ccr. This is the case if doa ≥ doat. doa and doat are values between 0 and 1.

This decision is particularly relevant in those cases in which the candidate list does not contain all elements that the best match comprises. (Otherwise doa = 1 and then also doa ≥ doat.) If doa ≥ doat and doat < 1 the missing elements of the best match are also inserted into the ccr by construction. Since they were not actually perceived, they are marked expected. A detailed account of these computations will be given in chapter [10].

Length of Traverse Buffer (lotb). The parameter lotb determines the length of the traverse buffer, ie the number of elements that can be stored in the traverse buffer. The traverse buffer is an increment buffer that contains pointers to the events that were selected for verbalisation but for which pvm-generation did not yet generate a preverbal message. While the elements are in the traverse buffer, linearisation can reorder them. The minimal value of lotb is 1 (lotb ≥ 1), because only if an element is stored in the traverse buffer it is accessible for pvm-generation. (A buffer of length 0 cannot store anything.) If lotb = 1 no linearisation can take place, because it requires at least two elements. If selection appends an element to the traverse buffer while it is filled, the first element (the head of traverse buffer) is dropped.

* In the implementation the parallelism of the processes is simulated by the Mozart/Oz system in which nC is implemented. Thus, it uses simulated parallelism (concurrency). Because of this, nC does not have any influence on the scheduling of the processes.
Latency (\(\text{LT}\)). The latency is the time that an element is kept in the traverse buffer until it is taken out by \(\text{pvm}\)-generation. \(\text{LT}\) is realised in the following way: when selection inserts an element into the traverse buffer a time stamp \((t_s)\) of the current system time \((t_c)\) is saved with it. \(\text{pvm}\)-generation monitors the head of traverse buffer. If \(t_s+\text{LT} \geq t_c\) then the element is removed by \(\text{pvm}\)-generation, and a preverbal message is generated for it.

Activation Threshold (\(\text{AT}\)). The activation threshold is used in \(\text{pvm}\)-generation. After \(\text{pvm}\)-generation decided to verbalise a refO, designations from the refO are chosen that constitute an adequate verbalisation. Since a refO can have a large number of designations, constraints are used for deciding which designations are used in the preverbal message. One of these constraints checks the activation of designations \(a_d\), and if \(a_d \geq \text{AT}\) this constraint is not violated. Since this is only a constraint the designation may be chosen despite the violation of this constraint if it is required for completing a preverbal message, and it can also be ruled out by another constraint.

8.5 Pre-processing unit (\(\text{ppu}\)) and perceived entities (\(\text{pes}\))

Although there is no sharp boundary between perceptual and conceptual processing, cf, for example, [Barsalou 1999] 588, \(\text{inC}\) is a model of conceptual processing only. Hence, this is an idealisation. All perceptual pre-processing is done in a separate component, the pre-processing unit (\(\text{ppu}\)). The output of the \(\text{ppu}\) is the input to \(\text{inC}\) or, more precisely to construction. The increments generated by the \(\text{ppu}\) are called perceived entities (\(\text{pes}\)). In this section I will give a rough sketch of how \(\text{pes}\) are computed. The exact way this is done is beyond the scope of this investigation. A more detailed description for the domain of the online description of the generation of sketch maps for which \(\text{inC}\) was first developed (see also section 14.3) is given in [Guhe & Huber 1999]. The component for motion events is still under development.

\(\text{pes}\) are computed from spatio-temporal coordinates. For motion events the input to the \(\text{ppu}\) consists of sequences of 5-tuples of the following form:

(1) \((\text{frameID}, \text{objectID}, \text{x-coordinate}, \text{y-coordinate}, \text{direction})\)

For example, a sequence of

(2) \([(1, 1, 50, 100, 90), (2, 1, 54, 112, 89), (3, 1, 52, 127, 91)]\)

of positions of an object, say, a plane, with the identifier 1 is translated into a \(\text{pe}\) representing a straight movement in the time interval \([1, 3]\) from coordinates \([50, 100]\)

* In the current version of \(\text{inC}\) the activation of a designation is static, ie it is assigned when the designation is created and does not change afterwards. However, this must be extended by a decay and reactivation mechanism, cf section 16.3.
8.5 Pre-processing Unit (PPU) and Perceived Entities (PES)

to \((52, 127)\). The direction of the object is the most recent one, i.e. 91 in this example. As can be seen in this (made up) example, the PPU must be able to cope with noise: the object is not moving along a completely straight path. It starts with a value on the x-axis of 50, at the next snapshot is has a value of 54, and at the final snapshot one of 52. Nevertheless, on the conceptual level it is desirable to consider this movement as straight. For this example a triple of PEs is generated that corresponds to the following simple referential net:

\[
\begin{array}{c}
\text{object} \quad r_1 \quad \eta \text{plane}(x) \\
\text{plane} \quad \eta \text{chpos}(r_2, x, r_3) \\
\text{situation} \quad r_2 \quad \eta \text{chpos}(x, r_1, r_3) \\
\text{at_time}(1, 3) \\
\text{path} \quad r_3 \quad \eta \text{chpos}(r_2, r_1, x) \\
\text{straight}(x) \\
\end{array}
\]

The exact format of PEs is not important here but is given in appendix B. The direction information is currently not used. The at_time attribute that is not part of the specification in chapter 6 only serves to synchronise INC to the points in time when the events occurred. This is necessary, because INC reads its input from a file, cf chapter 14. Note that the number identifying the object in the 5-tuples above is a different number than the one given in the pe attribute. The former is generated by the program with which the planes are displayed on the screen and enables the PPU to identify the objects, the latter is used on the interface between PPU and INC for referring to PEs, e.g. if the PPU must inform INC about an update of a PE.

Consider the case that the third 5-tuple is \((3, 1, 56, 127, 91)\). The PPU may still classify the movement as straight. Yet, since the PPU, just like INC, must work incrementally (otherwise INC could not operate incrementally), at a given point in time not all information is available that may be necessary for the correct and complete computation of an output. If the next 5-tuple is \((4, 1, 60, 142, 90)\) the path should be classified as curved, because the evidence for a curved movement is now much better, while the justification for assuming a straight movement got weaker. Since the increase of the value on the x-axis started in time frame 1, the curved movement does not start at (the current) time frame 4 but at frame 1. This means that corresponding backward corrections must be made, which is done by updating PEs.

The PPU segments movements according to two criteria:

1. an object changes between moving and not moving, and
2. the shape of the path changes.

An example of the first case is given in the introductory example. When the plane
stops (at the gate) the situation in which it moved ends, and a new situation starts in which it is not moving but standing. An example of the second case is that a plane that was moving on a straight path begins to follow a curved path. Then, the point where the plane commences the curved movement is the segmentation point between the two situations, cf the example discussed in section 14.2.

I assume another idealisation apart from the sharp distinction between perception and conceptualisation, viz the unidirectionality of information flow. This means, no information is given back from \text{IN}C to the \text{PPU}. This limits the cognitive adequacy of the proposed model, because there are clear influences from cognition (which includes conceptualisation) to perception. For example, priming effects do not only occur on a purely perceptual level but can also originate on the cognitive level. Cast in the \text{IN}C terminology this means two things. Firstly, expected elements in the conceptual representation are processed faster than non-expected elements. Secondly, the \text{PPU} must be sensitive of results of computations of \text{IN}C. Therefore, \text{IN}C should be extended to give feedback to the \text{PPU}. However, as I already said at the outset, I concentrate on the data-driven aspects of conceptualisation. Hence, there is no feedback to the \text{PPU} generated by \text{IN}C. Support that this is a valid idealisation comes from Barsalou (1999: 588f). He emphasises the fact that although the interrelation between perception and cognition is bidirectional, the bottom-up information, which I call data-driven information, in most cases dominates over top-down information. That is, information from perception to cognition exerts more influence than information flowing in the opposite direction.

A final remark on the term perceived entity. I chose this term – especially in contrast to basic entity, cf also the footnote on page 47 – to make clear that these entities are actually perceived and do not belong to a fixed level of granularity. For example, a motion event can be perceived as consisting of sub-events or as a whole. \text{IN}C is designed in a way that it can deal with both, ie it is able to segment one motion event into sub-events on the basis of conceptual knowledge, and it is able to group motion events to more complex events. Currently, however, only the second possibility is implemented.
CURRENT CONCEPTUAL REPRESENTATION (CCR)

The current conceptual representation was already introduced in the previous chapter as INC’s central storage device. Together with the concept storage (CS) it constitutes INC’s model knowledge. This chapter is devoted to describe its functions in more detail. The first section contains an account of the CCR proper. The other two sections describe the function of two special structures within the CCR, the traverse buffer and the traverse.

The CCR is a shared memory that contains INC’s knowledge of the current external state of affairs. Hence, it can also be regarded as blackboard. However, since blackboards can also be used as message passing devices, I prefer to call them shared memories, because the CCR is an integrated representation used by all processes of INC’s cascade.

9.1 The representation

The CCR contains the conceptual representation for a perceived state of affairs represented as a referential net. The knowledge on how to construct the conceptual representation is stored in the CS, cf section 6.2. The name current conceptual representation emphasises the fact that the content of the CCR is permanently changing. As I already laid out in section 8.2, its function is comparable to that of working memories of unified cognitive architectures.

It is important to keep in mind that the CCR is an internal representation, cf section 4.2. This means on the one hand that it is no one-to-one representation of the external world but contains some entities (more precisely: representations of entities) that are not present in the external world, eg expectations. On the other hand, it lacks entities that are present in the external world but did not ‘make it’ into the CCR, eg because they were not attended to, or they were filtered out in previous processing steps, eg in the PPU. Examples of actual representations are given and discussed in the context of how they are processed, cf the referential nets given in figures 4.2 and 14.3.

The basis of the representation in the CCR are the input increments of construction. In other words: each PE received by construction is added to the CCR. Based
on the pes the ccr is built up in an interplay between pes, rules in the cs, and the current state of the ccr by the construction process with the help of the concept matcher. Each time the ccr is modified by construction the selection process evaluates whether the changed element is verbalised, whether an utterance describing it is generated, and linearisation checks whether the elements selected for verbalisation should be reordered. pvm-generation uses the knowledge in the ccr for the generation of incremental preverbal messages. All knowledge that is part of a preverbal message is also stored in the traverse as memory of what has been said, cf section 0.3 and chapter 13. This knowledge can be used when following preverbal messages are generated, eg in order to generate reduced or anaphoric expressions for referring to objects.

In the implementation of nC the ccr is encapsulated in a separate process. This process is connected to the processes of nC’s cascade via increment streams. This means, there are two increment streams between the ccr process and each of the processes of the cascade, because increment streams are unidirectional. The increment streams are polled in an infinite loop by the ccr process. In this way nC ascertains that each process accessing the ccr has its particular view on the representation, ie that each process has access only to part of the representation. This is realised by means of restricted access functions that the ccr process executes on the referential net, depending on the cascade process that requested the operation.

There are three main motivations for using only one data structure:

1. It is more efficient in terms of storage capacity, because it reduces unnecessary redundancies.
2. It is compliant with the principles of unified cognitive architectures, cf section 7.6. One reason for this is that – especially in the more central components of cognition – the informational encapsulation is not strict, cf the discussion of quasi-modules on page 35.
3. It makes possible the use of indirect feedback, which in nC is mainly used in the traverse buffer, see below. Other possibilities, which are not used yet, include that selection and linearisation can make their choices dependent on the verbalisation refs, which are generated by pvm-generation and stored in the traverse.

If these reasons are not imperative, the internal conceptual representation needs not be organised in this way. It would be equally possible to give each process an internal storage, which would, however, require to convert the indirect feedback to explicit feedback. The disadvantage of organising the knowledge in this way would significantly increase the cost for maintaining, updating, and keeping consistent these storages – if consistency is required.

The ccr is no unstructured whole. As was already repeatedly indicated it contains two special sub-structures, the traverse and the traverse buffer. A symbolic depiction of these representations can be found in figure 9.1.

* In contrast to the definitions given here the traverse buffer has originally been considered a part of
9.1 The Representation

Fig. 9.1: The CCR including traverse and traverse buffer

**Term 9.1 Traverse.** The traverse is a path through the CCR, connecting all verbalisation refOs in the order in which they were generated. Since it contains all information that has been verbalised, it serves as basis of the discourse memory.

Verbalisation refOs are generated by PVM-generation for (non-verbalisation) refOs in the CCR. Each time it sends a refO to the formulator as part of an incremental preverbal message it creates a new verbalisation refO, which is also appended to the traverse. The verbalisation refO contains all information given to the formulator in the current preverbal message (term 13.2 on page 193). Additionally, the sequence in which the verbalisation refOs were generated is preserved.

**Term 9.2 Traverse buffer.** The traverse buffer is an increment buffer that contains pointers to all refOs that were selected for verbalisation but are not yet verbalised. If a refO is appended to the traverse buffer when it is filled, the first element (the head of traverse buffer) is dropped.

Each time a refO is verbalised, i.e., a verbalisation refO is generated for it, the pointer is removed from the traverse buffer. Since only the head of traverse buffer is accessible to PVM-generation, it is the only pointer that PVM-generation can remove. Otherwise, only selection can add or remove the elements stored in the traverse buffer by appending and replacing elements. (Linearisation can only change the order of elements.)

This was fine for the case that event and object structures were more or less isomorphic, as was the case with sketch generation events (Guhe & Habel 2001). This view causes problems when they are not. Additionally, the traverse was considered simply a path through the CCR. A severe problem with this is that this information structure is by no means rich enough for a discourse memory, because, for example, different verbalisations of different states of an object or event are not preserved and cannot be referred to. Among other things this makes it impossible to determine whether an entity can be referred to by a reduced referring expression like a pronoun.
9.2 Traverse buffer

The traverse buffer is an increment buffer. Its length is specified by the parameter $\text{l}o\text{tb}$, cf section 6.4. It contains pointers to refOs in the $\text{CCR}$ that the selection process chose for verbalisation but for which $\text{pvm}$-generation did not yet produce a preverbal message with the referenced refO as its starting point, cf Guhe (in print) and chapter 13. Thus, it contains only pointers to refOs, not copies of refOs like the traverse does. Pointers are used, because $\text{pvm}$-generation must use the current knowledge stored with a refO when it generates a preverbal message. The reason is that $\text{nC}$ is designed to generate verbalisations for the current state of its knowledge. Additionally, it would make little sense to use old versions of situation refOs that are stored in the traverse buffer, while all other refOs that are used in the incremental preverbal message are used with the knowledge that they contain at that point of time. (Updating the refOs in the traverse buffer each time they change in the $\text{CCR}$ would be possible but unnecessarily complex.)

The content of the traverse buffer can be changed by the operations defined in section 6.2, $R_\text{AppendToBuffer}$, $R_\text{Replace}$ (by selection), $R_\text{Reorder}$ (by linearisation), and $R_\text{Fetch}$ (by $\text{pvm}$-generation). They are symbolically depicted in figure 9.2. Since the traverse buffer contains pointers to refOs, elements need not be updated; the changes to the refOs are performed directly within the $\text{CCR}$.

Since the number of elements is limited, an append operation can cause the loss of an element, cf the definition of $R_\text{AppendToBuffer}$ on page 134. If selection appends an element when the buffer is filled, the first element – the head of traverse buffer – is deleted and the new element is appended. Observe that this need not be the element that is in the buffer for the longest time, because linearisation may put an element in head position that is newer; thus, even the most recent element can be pushed out of the buffer. Since selection has no access to the fill level of the traverse buffer, $\text{nC}$ does not ‘notice’ that the head of traverse buffer gets lost. (It is just recorded in the log file during a simulation.) While on the one hand this has some cognitive justification – we do not notice that we forget something –, selection could make its choices dependent on the number of elements in the traverse buffer:

For cognitive adequacy it might be better to delete the element with the lowest activation. However, since activation is not yet fully integrated in the $\text{nC}$ model, this is an idea for future versions.
if the buffer is in danger of being filled, selection can adapt its strategy accordingly.

The length of the traverse buffer can be considered proportional to the size of the part of working memory that is available for this temporary storage function. Yet, since the ccr itself fulfills the function of working memory, this is by no means a claim that the working memory has a capacity of the number of items specified by \texttt{LOTB}. (Perhaps even like the 'magical number seven plus or minus two' of [Miller, 1956].) Instead, \texttt{LOTB} is only a relative measure of the memory size for this temporary storage function in one simulation is in contrast to another simulation.

The second parameter relevant for the traverse buffer is \texttt{IT} (latency), cf section \texttt{8.4}. It determines how long an element must remain in the traverse buffer before it can be taken out by \texttt{pvm}-generation. This gives linearisation time to make its computations and reorderings, and it allows selection to revert choices. (Due to Extended Wundt’s Principle there are quite a lot of revisions.) It is not ascertained, however, that directly after \texttt{IT} has expired the element is taken out of the traverse buffer. Since \texttt{pvm}-generation only has access to the head of traverse buffer and elements can be reordered by linearisation, an element can be in the traverse buffer as long as \texttt{INC} is running – provided it never becomes head of traverse buffer after its latency expired.

Something that \texttt{INC} does not keep track of is that a speaker may remember wanting to say something, without actually doing so. Since the traverse buffer is only a temporary memory, such information is lost. There certainly are settings in which retaining such cases would be useful or even necessary. However, my solution is in accordance with the view that working memory is not capable of storing information permanently [Eysenck & Keane, 1995:129ff].

9.3 Traverse

The traverse is a path through the ccr that contains all verbalisation refOs in the temporal order in which they were created (term \texttt{9.3}). Thus, it contains all information \texttt{INC} sent to the formulator. In this way the traverse serves as a basic discourse memory. However, it must be kept separate from another part of the discourse memory. By recording what is sent to the formulator the speaker/conceptualiser keeps track of what he wants to say; by recording what comes in from the language comprehension system via the monitoring device he knows what he actually said. Since I do not consider information stemming from other interlocutors, these are the only kinds of information available. The traverse contains only refOs of the first kind, ie the traverse is that part of the discourse memory with the information of what the speaker intended to say. The information of what he actually said is represented in the monitored traverse (term \texttt{35.1} on page \texttt{227}), where the information received by the monitor is stored, cf chapter \texttt{15}.

When \texttt{pvm}-generation hands on a refO to the formulator, it has to decide which knowledge from the refO is used for the utterance that is currently being generated, ie it has to choose designations and/or attributes. At the point of time the head
of traverse buffer is deleted, a verbalisation refO for the refO is created, and the verbalisation refO is appended to the traverse. This verbalisation refO contains the information that is used in this particular verbalisation of the original refO.

The information of a refO used in a verbalisation is not simply duplicated. The references of a verbalisation refO to other refOs always refer to other verbalisation refOs, not to the original refO. This is necessary for using the traverse as discourse memory. Consider a short, abstract example. The ccr shall contain the following two refOs:

(1) \[\text{sort}_1 \quad \text{r}5 \quad '\text{NAME}' \]
\[\quad \eta x \ p_1(x)\]
\[\text{sort}_2 \quad \text{r}7 \quad \eta x \ p_2(x, r5)\]

A sequence of verbalisation refOs generated by pvm-generation out of this ccr may look as follows:

(2) \[\text{sort}_1 \quad \text{v}1 \quad '\text{NAME}' \]
\[\quad \text{verb}_\text{of}(r5)\]
\[\text{sort}_2 \quad \text{v}2 \quad \eta x \ p_2(x, v1) \]
\[\quad \text{verb}_\text{of}(r7)\]
\[\text{sort}_2 \quad \text{v}3 \quad \eta x \ p_2(x, v4) \]
\[\quad \text{verb}_\text{of}(r7)\]
\[\text{sort}_1 \quad \text{v}4 \quad \eta x \ p_1(x) \]
\[\quad \text{verb}_\text{of}(r5)\]

There are two verbalisations of r5 (v1 and v4) and two verbalisations of r7 (v2 and v3). In the first verbalisation of r5 the 'NAME' designation is used, in the second the description \(\eta x \ p_1(x)\). The important point here are the two verbalisations of r7. Both use the same description, \(\eta x \ p_1(x, r5)\). In order for the system to know, for example, which verbalisation of r5 the description belonging to v3 refers to, the original reference to r5 is replaced by the corresponding verbalisation refO (v4) during the generation of the incremental preverbal message. Using this method implies that one has to use (and resolve) forward references. At the point of time when v3 is created by pvm-generation and \(\eta x \ p_2(x, r5)\) is chosen to verbalise the refO, the reference to r5 must already be replaced by the corresponding verbalisation refO.
The reason is that immediately after its creation the verbalisation refO ($v_3$ in this case) is sent to the formulator. Yet, in the way $\text{PRM}$-generation operates this is no problem, because deciding upon one designation means that all refOs the designation refers to must be verbalised as well. For the example this means that another refO for the verbalisation of $\tau_5$ must be created as soon as $\text{PRM}$-generation decides to use $\eta_1 p_2(x, \tau_5)$ for verbalising $\tau_7$. This happens during the creation of $v_3$. The automatic refO numbering mechanism decides at this point to call the verbalisation refO $v_4$. If no designation is found for verbalising $\tau_5$ later on, $v_4$ will be generated as an empty verbalisation refO, i.e., a verbalisation refO containing no designation. The underlying idea is that such verbalisation refOs may be encoded as pro-forms by the formulator, e.g., as pronouns. See chapter 13 for further details.
CONSTRUCTION

Construction is the first of INC’s cascaded processes. The task it carries out is building up the ccr. In order to do so it reads pes from the ppu, calls the concept matcher to find more complex concepts that subsume the pes, and sends a notification to selection each time it modifies the ccr. Since the concept matcher interacts exclusively with construction, it could also be modelled as part of the construction task. Due to this close connection I will describe the concept matcher and the cs in this chapter as well.

In the ensuing chapters I will not provide Z-specifications for the algorithms, because as useful as these specifications proved to be for the clarification of theoretical concepts, they would require a lot of discussions of details. And these details would not serve the overall goal, viz to describe the functioning of INC. Instead, I will use a more abstract notation in pseudo-code in the following chapters. I assume the reader is familiar with such notations, and I will presuppose a set of common operations – especially list operations – like Append*, Filter, Replace, Delete, etc. These operations are important, because the Z-sequences for increment streams and increment buffers are implemented as lists.

10.1 The process

The construction process receives [perceived entities] (pes) and builds up the ccr from them.† In order to do so it is supported by the concept matcher, which is given a list of refOs, called candidate list. A candidate list contains pes and other refOs of the ccr, which were introduced into the ccr as results of previous matches. The concept matcher returns the best match, viz the concept in the concept storage

* Append will stand for both, appending a list to a list as well as appending an element to a list. Furthermore, I will not distinguish the Append operation on an increment buffer, which includes the deletion of an element if the buffer is full, from the Append operation on an increment stream, where this restriction does not exist. In the actual implementation these operations are, of course, realised by different functions. Mutatis mutandis this holds for the other functions too.
† More precisely: the basis of the ccr consists of refOs that are generated by construction when it receives the corresponding pes. I will simply refer to these refOs as pes.
VAR pe, #the PE received from the PPU
    fe, #focused element
    candidate_list, #candidate list (local context)
    best_match, match_list, doa #return values of the concept matcher
    doat #the value of the parameter DOAT

WAIT input_stream ≠ ⟨⟩
fe ← pe ← Fetch(input_stream)
Insert(pe, ccr)
REPEAT
    candidate_list ← ComputeCandidateList(fe)
    best_match ← (match_list, doa) ←
        ConceptMatcher(fe, candidate_list)
    IF doa ≥ doat THEN
        Update(match_list, ccr)
    ENDIF
    IF input_stream = ⟨⟩ THEN
        fe ← ComputeNewFE(best_match, fe)
    ELSE
        fe ← pe ← Fetch(input_stream)
        IF Type(pe) = new THEN
            Insert(pe, ccr)
        ELSIF Type(pe) = update THEN
            Update(pe, ccr)
        ENDIF
    ENDIF
UNTIL pe = eof

Figure 10.1: The construction process

(cs) that has the highest degree of agreement (DOA) with the refOs in the candidate list. If the DOA is greater than or equal to the degree of agreement threshold (DOAT) construction updates the ccr accordingly; otherwise the match is ignored.

Since the concept matcher only interacts with construction and the cs is only accessible by the concept matcher, both could also be seen as part of construction. However, the cs represents nC’s long-term memory, ie the knowledge that is available independently from the currently observed scene. Thus, the concept matcher constitutes the interface between working memory and long-term memory. Viewing the concept matcher as separate process helps to keep these memories apart.

Construction is the process of nC behaving most similar to an anytime algorithm, because it tries to find more complex concepts until no new candidate list can be assembled, or a new PE is available. In the first case it suspends itself until it receives a new PE. The pseudo-code describing the construction process is given in figure 10.1. Variables are written in small letters, functions with a capital
letter. Generally speaking, the incremental algorithm of construction (in the sense of term \[5.12\] on page \[95\]) consists of calling the concept matcher and evaluating the match result.

Construction waits until the first \(pe\) is available in the input stream (the wait statement). As soon as this is the case the \(pe\) is fetched and inserted into the \(ccr\). It is also the new focussed element (\(fe\)). Then construction loops until the \(ppu\) signals that no more input will follow by sending \(eof\). The first operation in the loop consists of taking \(fe\) and computing the candidate list for the call of the concept matcher. The function \(ComputeCandidateList\) is described below. So, the local contexts of construction are candidate lists.

The return value of the function calling the concept matcher is a pair consisting of \(match\_list\) and \(doa\), which is also called best_match. If the best_match is good enough, i.e., if \(doa \geq doat\), then the \(ccr\) is updated accordingly. The \(Update\) operation takes into account that refOs that are part of the best match can already be in the \(ccr\). There are three cases in which this operation is particularly critical. Firstly, the \(doa\) can be smaller than \(1\). This means that the match contains refOs for which there are no corresponding perceived ones in the \(ccr\). In this case new refOs are created and inserted into the \(ccr\) for the ones that have no counterpart in the \(match\_list\). These refOs are marked expected.

Secondly, the best match can be different from the previous one. For example, it may turn out that the \(plane\) that is expected to move towards \(gate\-\text{B21}\) is not moving there but, say, towards \(runway\-3\). In this case the expected refOs must be discarded, here the expected situation and path refOs.

Thirdly, there may already exist an expected refO in the \(ccr\) for a newly received \(pe\). This case is quite complex, because there are now two refOs for the new \(pe\): the expected refO and a refO representing the \(pe\). After the match the identity of these two refOs must be established with the help of the information in the best match, and the refOs must be fused. Informally, one could say that in the fusion the old refO changes its status from expected to regular, and the newly perceived entity contributes the information of how the expected (and now regular) refO actually 'looks'.

Figure \[10.2\] shows the five most important stages of the construction of the first part of the event structure for the example of chapter \[1\]. This is the part in which the \(plane\) moves towards the \(gate\) and stops, which corresponds to the first two phases of the scene. In stage \(1\) only the perceived entity \(move\_p\) is present, which

* Calling \(eof\) a perceived entity is a bit misleading, but it simplifies the model.
† Executing the loop can be made dependent on the fact whether the \(pe\) is already present in the \(ccr\) as an expected refO. If this is the case the status of the expected refO is changed to regular, see below. This way it would be possible to model a real priming effect, because expected refOs are integrated directly without calling the concept matcher, while removing an expectation requires extra effort.
‡ Since the available knowledge of the concept matcher consists of the \(cs\) and the candidate list, it has no third argument as is usually required for an incremental algorithm. In other words, the concept matcher cannot modify a representation that exists independently from it.
§ The match involving the \(pe\) must yield the same complex concept as the last one, because otherwise the expectations would have been discarded already.
is short for the refO representing the situation in which the plane moves. Assume that the best match resulting from the call of the concept matcher leads in stage 2 to the insertion of three expected refOs, depicted as the grey nodes and edges in the figure. In stage 3 the next pe is read by construction. This is the left black be_atp node. The concept matcher is then called with the two (black) concepts move_p and be_atp as candidate list. Note that before the call of the concept matcher the two left be_atp nodes are in no way related, i.e., the relation indicated by the bidirectional arrow is not present before the concept matcher is called. The resulting best match, given on the right side in stage 3, is the same as the previous one. With the help of the best match it can now be established that the two be_atp nodes must be identical and, therefore, fused. The result of the fusion results in stage 4. Finally, since all concepts for the match are present (doa = 1), the status of the other two expected nodes changes to regular, shown as stage 5. cs rules and the functioning of the concept matcher are discussed below. The representation in this example only includes the event structure. Similar computations have to be carried out for

* The identity of the move_p in the ccr and the move_p of the best match must also be established again. The corresponding arrow is left out in the figure for better readability.
10.1 THE PROCESS

the sub-structures containing the spatial and object information.

After the ccr is updated the next call of the concept matcher is prepared. The first step is to check whether the input_stream is empty. In this case the new fe is computed from the old fe and the best_match. The one possibility is that the new fe is the old one. Then a different local context must be determined before the concept matcher is called again. The other possibility is that it is an element of the best match. In particular, after a successful match, ie if doa \( \geq \text{doar} \), the new focussed element is a complex concept of the best match. Since the concept matcher can give back multiple complex concepts in a match, cf section 10.3 the new focussed element in these cases always is the situation refO. (If multiple complex concepts are computed by the concept matcher one concept always is a situation refO.) The reason for making this simplification is, again, that I investigate event conceptualisation.

If a new input increment is present in the input_stream it is fetched and becomes the new focussed element of construction. If it is a new pe it is inserted into the ccr; if it is an update increment the old pe that is already in the ccr is updated. If also pe \( \neq \text{eof} \) the next iteration of the loop starts with the computation of the candidate_list.

The part of construction handling the suspension of the process is not given here for reasons of brevity. The process considers suspending itself after the candidate_list of the next iteration is computed. If the values of best_match, fe, and candidate_list are identical to the previous ones, construction suspends itself until a new pe is available in the input_stream. Furthermore, the notification sent to selection is not given here as well. Each time construction modifies the ccr, eg by inserting a new refO, it tells selection that a modification has occurred and what the modified refO is.

The function computing the candidate lists is given in figure 10.3. It depends on whether the focussed element (fe) is of the sort situation, object, or spatial_entity. In the first case the largest local contexts are generated, which is due to the central role that situation refOs play in the domain and the representations used here. Informally speaking: these refOs are the glue of the conceptual representation. The designations of the fe are searched for those that have a predicate linking the situation refO to other refOs participating in the situation. The predicates of these designations are mainly chpos, be_at, start, and stop. Additionally, the situation refOs directly connected to fe by a temporal relation, cf section 4.5 become part of

* This is currently not done by nC, because construction can deal with all examples investigated so far without determining different local contexts. Consequently, the heuristic computing candidate lists (ComputeCandidateList) does not do this. It is necessary in the general case, though, because the optimal candidate list that yields the highest doa with an entry of the cs may not have been found in the first attempt. The idea of different, more precisely: increasingly broadening local contexts originated in the analysis of drawings of complex sketch maps consisting of up to 150 elements, which corresponds roughly to the number of res for events and for objects (300 in toto).

† This is the function for motion events; the one for sketch maps looks slightly different but is basically the same.
fun ComputeCandidateList(fe)
    if Sort(fe) = situation then
        return Append(
            ExtractRefOs(Filter(DescriptionPreds(fe),
                Member([chpos, be_at, ...])),
            ExtractRefOs(Filter(AttributeNames(fe),
                Member([temp_rels])))
        )
    elseif Sort(fe) = object then
        return nil  # currently, objects are not matched
    elseif Sort(fe) = spatial_entity then
        return ExtractRefOs(Filter(DescriptionPreds(fe),
            Member([finalpoint, startpoint, ...]))
    endif
endfun

Figure 10.3: Construction’s heuristic for establishing the candidate list

the local context.

In the second case, ie if the sort of fe is object no matching takes place, because the structure of objects is only of peripheral interest in this investigation. An example where this may be of interest is if two objects are moving on parallel paths. Then, they should be grouped. (But even in this case the grouping is due to the parallelism of the paths, which is not represented with the objects.) Matching of objects is more important in the domain of sketch maps [Guhe & Huber 2000].

The third case, finally, captures refOs of sort spatial_entity. In particular paths and locations can be grouped with calls of the concept matcher with local contexts computed this way.

10.2 Concept storage (cs)

The concept storage (cs) is inC’s second knowledge representation besides the ccr, but in contrast to the ccr it does not change over time. In particular, nothing is added, which would correspond to the learning of categories. The cs corresponds to the long-term memory of inC, and is exclusively accessible by the concept matcher, which has full (read-only) access. Hence, there are no different views on the cs. Like the ccr the cs is a referential net. It consists of production-like rules of how to infer more complex concepts from simpler ones (and vice versa). The rules partition the cs, ie they are not interconnected. A very simple rule for connecting two paths is as follows:

```
(1)   path    r1   ix endpoint(r3, x)
     part_of([r4])
```

176
In this rule \( r_1 \) and \( r_2 \) represent two paths that meet at the location represented by \( r_3 \). The concat attribute of \( r_4 \) is a special parts attribute, which not only says that the listed refOs are parts of the refO but also that the subsumed refOs are concatenated in the sequence in which they are given in the list. Thus, in this rule the refOs \( r_1 \)–\( r_3 \) are matched onto a complex path, \( r_4 \). The attribute \( \text{no\_of\_refOs} \) specifies the number of refOs that must be present for a \( \text{doa} = 1 \). The concept matcher computes the \( \text{doa} \) as ratio of the number of refOs present to the number of refOs specified in the rule, cf. equation \( \text{equation} \) on page \( \text{page} \). In this rule two paths and the location connecting them must be present; thus, the \( \text{no\_of\_refOs} \) is three.

In a more detailed version of this rule the paths each contain a description of their form.

\[
\begin{align*}
\text{path} & \rightarrow r_2 \quad \text{ix startpoint}(r_3, x) \\
\text{part\_of}(\{r_4\}) & \rightarrow \\
\text{location} & \rightarrow r_3 \quad \text{ix endpoint}(x, r_1) \quad \text{ix startpoint}(x, r_2) \\
\text{path} & \rightarrow r_4 \\
\text{concat}(\{r_1, r_2\}) & \rightarrow \text{no\_of\_refOs}(3)
\end{align*}
\]

This rule takes two refOs representing a straight path \( (r_1) \) and a curved path \( (r_2) \). The complex refO that is generated by applying this rule differs from the one above in that it contains an additional description specifying the form of the concatenated path \( (p\_curved) \) stands for partially curved) and in that the location where the two paths meet is a transition point. The special designation attribute \( * \) marks

\[
\begin{align*}
\text{path} & \rightarrow r_1 \quad \text{ix endpoint}(r_3, x) & \eta x \text{straight}(x) \\
\text{part\_of}(\{r_4\}) & \rightarrow \\
\text{path} & \rightarrow r_2 \quad \text{ix startpoint}(r_3, x) & \eta x \text{curved}(x) \\
\text{part\_of}(\{r_4\}) & \rightarrow \\
\text{location} & \rightarrow r_3 \quad \text{ix endpoint}(x, r_1) \quad \text{ix startpoint}(x, r_2) & \eta x \text{transitionpoint}(x, r_4) \rightarrow * \\
\text{path} & \rightarrow r_4 & \eta x \text{p\_curved}(x) \\
\text{concat}(\{r_1, r_2\}) & \rightarrow \eta x \text{transitionpoint}(r_3, x) \\
\text{no\_of\_refOs}(3) & \rightarrow
\end{align*}
\]

* This attribute is no given in the formalisation in chapter 6.
those designations that have to be added to refOs already present in the ccr. So, if construction uses the match result of an application of this rule, it inserts a refO corresponding to \( r_4 \) into the ccr, say \( r_{20} \), and add the \( \iota x \ transitionpoint(x, r_{20}) \) description to the refO that has been matched onto \( r_3 \).

However, the rules given so far are only of illustrating character. Among other things their results consist of only one complex refO. The rule given in figure 10.4 is

In the definition on page 131 I said that no designation attributes are used in this investigation. This is the only exception. Since this designation attribute contains meta-knowledge, it is not regarded as part of the representation proper.
10.3 Concept matcher

The main function of the concept matcher is to perform categorisations \cite{Goldstone1994,Murphy2002}. In doing so it mainly uses features and feature vectors as they are described in \cite{Smith1995}, who presents a classical account of categorisation, concepts, and similarity. Thus, the concept matcher does not provide a new method of categorisation but more or less conforms to this classical view. However, there are two special points about the concept matcher. Firstly, it enables construction to generate expectations, i.e., it is capable of computing incomplete matches (categorisations). Secondly, the concept matcher is an \textit{n:m}-matcher. That means, it can find \textit{m} more complex concepts for \textit{n} simpler ones within one match.

The rules stored in the \textit{cs} can in principle also be used to find simpler concepts for more complex ones. In particular it is possible to segment a (complex) concept into multiple simpler ones. This is particularly interesting if the \textit{ppu} is not only capable of producing the simplest concepts as it is assumed in the following but also concepts of a higher granularity. Thus, the possibility of segmenting concepts is not used up to now. Furthermore, the refOs in the candidate list must be on the same level of granularity, i.e., they must not be related by \texttt{parts} or \texttt{part_of} relations.

Figure \ref{fig:concept_matcher} shows the main loop of the concept matcher process. After initial-

\begin{verbatim}
VAR candidate_list, #the candidate list received from construction
doa, #the degree of agreement of the best match
match_list #the match list given back to construction

REPEAT
  WAIT input_stream \neq \langle \rangle
  input_increment \leftarrow Fetch(input_stream)
  IF input_increment \neq terminate THEN
    candidate_list \leftarrow input_increment
    (doa, match_list) \leftarrow MatchCandidateList(candidate_list)
    Append((doa, match_list), outputStream)
  ENDIF
UNTIL input_increment = terminate
\end{verbatim}

Figure 10.5: The concept matcher process

actually used by \textit{incC} for connecting two \texttt{PATHS}, for instance in the worked example in section \ref{sec:worked_example}. This rule not only matches the two \texttt{PATHS} but also the bearer of motion, which must be an object – a \texttt{PLANE}, for example –, and the \texttt{SITUATIONS} in which the movements along the \texttt{PATHS} take place. These three kinds of refOs are bound together by the chpos descriptions.
fun MatchCandidateList(candidate_list)

var doa,

start_refOs, #list of refOs the matching starts with
match, #a match of the candidate list onto a rule of the cs
list_of_matches, #list of all matches

for all rs ∈ start_refOs do
for all rc ∈ candidate_list do
if InitialMatch(rs, rc) then
match ←
BuildFinalMatchList(
BuildPartsList(
BuildPartOfList(
BuildBasicMatchList(rs, rc)))))
doa ← ComputeDoA(match)
list_of_matches ← Append((match, doa), list_of_matches)
endif
endfor
endfor

return FindBestMatch(list_of_matches)
endfun

Figure 10.6: The function MatchCandidateList

isation it waits until a new input_increment is available in the input_stream. If the input_increment is the signal for termination the loop ends. Otherwise it is a candidate_list, for which the matching function MatchCandidateList is called. This is the incremental algorithm of the concept matcher. It returns the best match, which is a pair consisting of the match_list of refO pairs (those consisting of a ccr and a cs refO) and the corresponding doa of the match. The best match is appended to the outputStream.

As can be seen the concept matcher has no access to the ccr, ie the available knowledge of the concept matcher’s incremental algorithm consists of the cs and the candidate list it obtained from construction. Thus, the concept matcher does not have the ability to change available knowledge of any kind.

The function MatchCandidateList is shown in figure 10.6. It takes the candidate_list sent by construction and searches the cs for the best matching concept, the best match. The function consists of two nested for-loops. The outer loop runs over all refOs that are in the start_refOs list, the inner one over the refOs in the candidate_list. Each rule in the cs has start_refOs with which matching can be started. For example, the rule in figure 10.4 has three such refOs, r1, r2, r3 (not indicated in the figure). Using start_refOs has the effect that the concept matcher does not have to try all possible combinations of refOs in the rule and refOs in the candidate_list. This increases efficiency in particular if the rule is symmetric,
eg a rule that groups together two planes moving in parallel, where it makes no
difference whether planeA is matched onto rₓ of the cs rule and planeB onto rᵧ
or vice versa.

All refO's in the candidate list are compared whether they can be matched onto
one of the start_refO's. Only if the InitialMatch is possible the doa for the rule
in computed. If the same rule can be applied to a candidate list more than once, the
match with the highest doa is taken.

The matching proper is done in four distinct steps. Before I describe these steps,
however, let me first give some examples of how the doa is determined. Generally
speaking, the doa can be determined in many different ways. Since measuring the
similarity of concepts is not in the focus of the present work, I simply use the fol-
lowing equation here:

\[ \text{doa} = \frac{\text{number refO's present}}{\text{number of refO's required by rule}} \]

A better similarity measure may be required in the future, eg one that also accounts
for how similar a ccr refO is to the corresponding cs refO, but for the present
purposes of the model this one does suffice.

Now let us look at some examples. The rule (1) given in section 10.3 requires
three refO's in the candidate list that correspond to r₁–r₃ in order to yield a doa = 1.
That is, the doa of this rule is computed by doa = x/3 where x is the number of
matching refO's in the candidate list. Consider that this rule is used for matching
the following two candidate lists, each consisting of only one element

(4) spatial_entity — r₃₀

(5) path — r₃₁

In both cases doa = 0, because none of the simpler refO's in the rule (r₁–r₃)
can be matched onto the one in the candidate lists. The reason in the first case is
that not even the sorts of the refO's match – spatial_entity is not specific enough.
In the second case r₃₁ has no common designation with one of the cs refO's, which
is the minimal condition for two refO's to be matched onto each other.

For the following candidate list doa = 1/3.

(6) path — r₃₂ — ix endpoint(r₄₀, x)

Here, the refO in the candidate list and the cs refO have one designation in com-
mon. Thus, r₃₂ can be matched onto r₁. Observe, however, that this is a very
strange candidate list, because it contains no corresponding refO for r₄₀ / r₃. Hence,
construction should not produce such a list, and the actual implementation of con-
struction does not do it.

Finally, the candidate list
yields \( \theta = \frac{2}{3} \). The corresponding match list is

\[ ((\text{ccr}(r_{32}), \text{cs}(r_1)), (\text{ccr}(r_{40}), \text{cs}(r_3))) \]

This means that \( r_{32} \) of the ccr is matched onto \( r_1 \) of the cs, and \( r_{40} \) onto \( r_3 \). It is also the basic match list of this example, the first of the four steps in computing a match, cf figure 6.6. Computing a match consists of a sequence of building up the following four lists:

1. basic match list,
2. part-of list,
3. parts list,
4. final match list.

The first step in constructing a basic match list is to find an initial match. As I already mentioned above this is done by the function \text{InitialMatch}, which compares the start_refOs to the refOs in the candidate list. Due to the two nested for-loops all permutations are tried, ie all refO\'s of the candidate list are matched onto all start_refOs of the cs rule. According to the initial match the other refOs in the candidate list are matched onto refO\'s of the rule so that each refO of the candidate list is matched onto a refO of the cs rule.

With the basic match list no hierarchical relations have been established yet. The next step, therefore, is to construct the part-of list. It contains the possible complex refO\'s for each refO in the basic match list. This is necessary if a simpler refO is part of multiple refO\’s. This case is rather rare in the domain of motion events, but it is quite frequent in the domain of sketch maps, where, for instance, one line can belong to multiple objects. In those cases the parts lists contain more than one element. For the example at hand the part-of list is

\[ ((\text{complexes}([r_4]), \text{part}((\text{ccr}(r_{32}), \text{cs}(r_1))), \text{complexes}([r_4]), \text{part}((\text{ccr}(r_{40}), \text{cs}(r_3))))]) \]

From the part-of list the parts list is constructed. One can imagine this list as part-of list upside down: for each complex refO in the part-of list, ie each refO in the parts lists, its parts are determined. For the example it is a list of only one element:

\[ ([\text{complex}(r_4), \text{parts}([\text{ccr}(r_{32}), \text{cs}(r_1)), \text{ccr}(r_{40}), \text{cs}(r_3)))]]) \]

An issue not further discussed here is that the match must, one might say of course, be consistent. That is, if \( r_{40} \) of the \( x \) endpoint(\( r_{40}, x \)) description is matched onto \( r_3 \) of the cs rule, this match must hold in all cases. If another refO contains a description \( \eta x \ p(x, r_{40}) \) this occurrence of \( r_{40} \) must also be matched onto \( r_3 \).
This list is almost in the format that is given back to construction. The final match list is computed from the parts list by extending each pair in the parts list to a triple. For this the information about the refOs that are still missing for a complete match are added. Here:

(11) \[ \text{complex}(r_4), \text{parts}(\text{complex}(r_32), \text{cs}(r_1)), \text{complex}(r_40), \text{cs}(r_3)), \text{missing}(\text{r}_2)) \]

So, the final match list is the result of matching the candidate list onto one of the cs rules. For this list the doa is determined. Since the final match list can contain multiple complex concepts (complex-refOs), the doa of the list is computed as mean of the doa of each complex concept. For the example it is:

(12) \[ \text{doa} = \frac{\frac{1}{3}}{2} = \frac{2}{3} \approx .67 \]

For a more interesting example assume that for each of \( r_7 \) and \( r_8 \) in the rule given in figure [no.4] one part-refO was found. The doa for this final match list then is

(13) \[ \text{doa} = \frac{\frac{1}{2} + \frac{1}{2}}{2} = \frac{2}{2} = \frac{5}{12} \approx .42 \]

All of these matches (all final match lists) are collected together with their doa in the list_of_matches. If no match was found, the empty list (nil) with a doa of 0 is taken instead. After all rules in the cs have been tried, the match with the highest doa is chosen from the list_of_matches. This is the best match, which is the result of the function MatchCandidateList and given back to construction by the concept matcher. The final match list that is given back to construction as part of the best match is referred to as the match list in the rest of this investigation. If matching was not successful the best match consists of the match list nil with a doa of 0. If multiple matches with identical doa compete to be the best match the first match in the list_of_matches is given back to construction. Note that this is another case where this method suffices for the present purposes. Generally, there are better solutions. One would be to return all these matches. Construction then has to decide which one of these it takes as best match. Another one is to let construction pursue multiple possibilities in parallel. A third one to ascertain that this case cannot occur, e.g by introducing a decision function that can be made dependent on further factors.
Selection evaluates each element (refO) of the ccr that is added or changed by construction. As can be seen from the previous chapter, the information that is processed by construction mostly is information that is perceptually prominent. It can, therefore, be safely assumed that the increment construction passes on to selection is in the focus of attention. Either it is information that was just received from the PPU or created with the help of the concept matcher. That this approach is cognitively adequate is shown in Guhe & Habel (2001), in Guhe, Habel & Tschander (2003b), and in section 14.3, where Inc is tested against empirical data.

The outer loop of selection is shown in figure 11.1. Each time the construction process changes a refO in the ccr it notifies selection. The loop runs until construction signals that it terminates, in which case selection does so as well. This is done via a special increment that is sent by construction when it terminates. (Similarly selection sends such an increment to linearisation when it terminates, which is not shown in the figure.)

Each iteration of the loop waits until a new input increment is available, which is then fetched from the input_stream. Selection only considers situation refOs for verbalisation. Hence, input increments of another sort than situation are ignored.† The situation-refO can contain the information that

1. a new refO has been added to the ccr,
2. an existing refO has been updated, or
3. an expected refO has been discarded.

This limitation is, as already pointed out repeatedly, due to the fact that I am only concerned with event conceptualisation. In general, refOs of all sorts can be selected. The difference in selecting, say, an object refO is that the resulting utterance is about the object not the situation the object is in. In other words, the focus (topic) of the resulting preverbal message generated by PVM-generation is different.

† I make a simplification in these formulations. The increments between construction and selection – like the increments stored in the traverse buffer – are actually pointers to refOs in the ccr. Thus, the full formulation would be: input increments that are pointers to refOs in the ccr that have another sort than situation are ignored.
REPEAT
  wait input_stream \neq ()
  input_inc ← Fetch(input_stream)
  if input_inc \neq terminate THEN
    if Sort(input_inc) = situation THEN
      if Type(input_inc) = new or Type(input_inc) = update THEN
        MakeDecision(input_inc)
      elsif Type(input_inc) = expectation_discarded THEN
        Delete(input_inc, traverse_buffer)
      ENDIF
    ENDIF
  ENDIF
UNTIL input_inc = terminate

Figure 11.1: The selection process

FUN MakeDecision(input_inc)
  parts ← Simpler(input_inc, traverse_buffer)
  part_of ← Complex(input_inc, traverse_buffer)
  if parts \neq () THEN
    Replace(parts, input_inc)
  elsif part_of = () THEN
    Append(input_inc, traverse_buffer)
  ENDIF
ENDFUN

Figure 11.2: The standard selection algorithm

The third case occurs when an expectation no longer holds, e.g. because the concept matcher found a new best match for the elements under consideration. If there is a pointer to this refO in the traverse buffer it is deleted. The first two cases are treated identically up to now by calling MakeDecision, which decides whether the situation refO under consideration is selected. It differs depending on the selection strategy that is used. (The selection strategy is chosen by the user before 
\textsc{inC} is started, cf chapter 14.) 
\textsc{inC}'s standard selection strategy is to select the most complex situation refO for verbalisation that is currently not selected, cf figure 11.3. Thus, the decision is made with regard to the part-of hierarchy. There are three cases:

1. if a more complex refO is already selected, i.e. in the traverse buffer, nothing is done,
2. if simpler refOs are already selected, those refOs are replaced,
3. otherwise the refO is appended to the traverse buffer (this is the default).

In other words, the hierarchical relations between the refO under consideration and the refOs that are currently in the traverse buffer are evaluated. At first, it is
fun MakeDecision(input_inc)
    if traverse_buffer = ⟨⟩ then
        Append(input_inc, traverse_buffer)
    elsif ∃ e ∈ traverse_buffer AND
        Granularity(input_inc) = Granularity(e) then
        Append(input_inc, traverse_buffer)
    endif
endfun

Figure 11.3: An alternative selection algorithm

tested whether the refO under consideration has parts refOs in the traverse buffer, ie refOs that are lower in the part-of hierarchy.† If the refO has parts (simpler) refOs in the traverse buffer, then the first of these refOs, ie the headmost refO, is replaced by the refO under consideration; the others are deleted.‡ Both operations are carried out by only one operation, which is called Replace in figure 11.3. This is necessary, because otherwise inconsistencies might arise if linearisation tries to exchange two refOs in the traverse buffer while selection is just performing this operation.‡

After this, it is tested whether the refO under consideration is part of traverse buffer refOs. If this is not the case then it is appended to the traverse buffer.§ Thus, the three cases given above are condensed to just two cases in the implementation. The local context of this selection strategy is the traverse buffer. No other information from the ccr is used.

Each modification of the traverse buffer by the operations Replace, Append, or Delete includes a notification of the linearisation process by an output increment, which is not given in figure 11.3 for reasons of brevity.

An alternative selection strategy is given in figure 11.3. While in the standard strategy the level of granularity of the selected situation refOs is not the object of computation, this is the main point of the alternative selection strategy. Nevertheless, in this selection strategy the level of granularity is also not explicitly fixed in advance but a result of the computations of inC. This selection strategy uses the additional function Granularity, which identifies the level of granularity of a refO.¶

* The parts attributes are expanded as far as necessary; the underlying relation is transitive.
† This causes no gaps in the traverse buffer. The traverse buffer is defined as a sequence of increments. This means that it is implemented as a list, not as an array.
‡ In fact, in the implementation I was even more cautious in that the complete selection algorithm is encapsulated in a critical section so that such inconsistencies can never arise.
§ The Append operation ascertains that the head of traverse buffer is deleted if the maximal length of the traverse buffer has been reached before the operation is applied, cf also the definition of R_AppendToBuffer on page 136.
¶ Currently, it uses a rather simple method, viz using the part-of hierarchy to count the number of edges that the refO is away from the pss from which it is built up. This method fails, for example, if the number of edges to different pss is not the same. However, in the examples investigated so far this is not the case.
A refO is selected in two of three possible cases:

1. if the traverse buffer is empty, the refO is selected
2. if the refO has the same granularity as a refO in the traverse buffer the refO is selected as well
3. otherwise the refO is not selected

In other words, the effect of this selection strategy is that only refOs of the same granularity are present in the traverse buffer. A change of granularity can only occur if the traverse buffer is empty.
LINEARISATION

LINEARISATION, as I already pointed out on various occasions has been considered only scarcely and theoretically in the context of inC up to now. This means in particular that it is not implemented yet. The main reason for this is that due to the strictly sequential order of the input information only one event thread is conceptualised.

Term 12.1 Event thread. An event thread is a sequence of events that have the same bearer.

(The bearer of motion is the entity that is moving. It can be different from the actor, the entity that caused the motion of the moving entity, cf Eschenbach, Tschander Habel, & Kulik (2000).) Hence, a reordering of events is expendable for the moment. The only consequence is that no verbalisations are for generated that describe events in an order different from the one in which they occurred. Therefore, I will only point out some important issues and ideas that will have to be taken into account when the linearisation process is actually modelled and implemented.

First of all it must be stressed again that linearisation performs a linearisation of utterances (sub-intentions, situations, events) not of phrases. A method of how phrases can be linearised incrementally within a model of the formulator is Performance Grammar, described in Kempen (forthcoming), Kempen & Harbusch (2002, in print). This method depends strongly on the sequence in which the increments arrive in the formulator, i.e., the sequence in which pvm-generation produces output. Note that pvm-generation does not perform a linearisation task either.

Linearisation in the online description of events can be investigated by an extension of the setting used here in which the participants have to verbalise concurrent events, cf Guhe, Habel, & Tschander (in print) and section 13.3 for a first proposal. Concurrent means that in the scene the participants observe more than one event happening simultaneously. In the setting used here such scenes may consist of multiple planes moving simultaneously. A linearisation should be performed if selection alternatingly appends events of different event threads to the traverse buffer, say event threads A and B. In this case linearisation should reorder these events in such a fashion that not one event of thread A, then one of B, then again one of
A, then again one of B, etc is verbalised. Instead, some events belonging to event thread A should be verbalised, then some belonging to B, etc. This could be captured by a new parameter for $\text{trC}$, a event thread retention. See also chapter 16 on this idea.

Like the selection strategies described in chapter 11, linearisation should be able to work according to different linearisation strategies. One of these strategies can be to order the events according to their importance so that the most important event is mentioned first. The importance of an event can, for instance, be determined on the basis of its perceptual prominence. Another strategy might consist in uttering contrasting events first so that utterance (1) is preferred over (2), which would be preferred by the first strategy. The second linearisation strategy corresponds to selection strategies preferring contrasting information.

(1) Plane A and plane B move straight ahead. Plane A stops, and plane B goes on straight.
(2) Plane A and plane B move straight ahead. Plane B goes on straight, and plane A stops.

In section 3.2 I already argued that all deviations from the chronological order must be overtly marked. That means if the order in which events are verbalised is different from the one in which they occurred, this must be made explicit in the utterance. The task by which this has to be ascertained is $\text{PVM}$-generation. Thus, developing linearisation means that $\text{PVM}$-generation has to be extended as well.

* Since selection always decides whether an event is verbalised after it was introduced or updated by construction, such a ordering of events in the traverse buffer is quite likely, at least if the event threads proceed with a similar pace.
PREVERBAL MESSAGE GENERATION

PVM-generation corresponds to Levelt’s [1989] microplanning, cf section 3.2. More accurately, one should speak of microselection, because there is no microlinearisation within the conceptualiser. This is the duty of the formulator. A model that takes care of microlinearisation is, for example, Kempen’s Performance Grammar [Kempen forthcoming, Kempen & Harbusch 2002 in print]. In the terminology of this investigation the selection of utterances, described as subintentions in section 3.2, is done by the selection process, while PVM-generation selects the means to express the utterance. This division of labour is analogous to the one between what-to-say and how-to-say that is used to distinguish conceptualiser and formulator [De Smedt, Horacek, & Zock 1996]. One might call it what-to-utter and how-to-utter.

The (precursors of the) ideas presented in this chapter were already previously published. Guhe, Habel, & Tappe [2000] and Guhe [in print] introduce and discuss the idea of incremental preverbal messages, which are the topic of section 13.1. Guhe, Habel, & Tschander [2003a,b] describe the incremental algorithm with which the incremental preverbal messages are generated. This work is presented in section 13.2. This chapter closes with considerations of how the referential nets approach can be exploited to establish co-referential relations between entities occurring in subsequent incremental preverbal messages in section 13.3.

13.1 Incremental preverbal messages

The term incremental preverbal message emphasises the incremental mode of generating preverbal messages.

Term 13.1 Incremental preverbal message. An incremental preverbal message is a preverbal message that is generated incrementally.

The preverbal messages produced this way are equivalent in their expressive power to (non-incremental) semantic structures proposed by linguists, especially those by
As I already pointed out Jackendofft’s semantic representations are on the level of abstractness that Levelt proposes when he speaks of preverbal message. These semantic structures are rather close to conceptual representations compared to other approaches, e.g., the one by Bierwisch & Schreuder (1992). In fact, they are regarded as special conceptual representations. Wiese’s (in print) account of how to get from a conceptual to a semantic representation works in a similar fashion and corresponds, therefore, to the task carried out by pvm-generation.

One implication from the incremental mode of generation is that an incremental preverbal message is complete only when a new one is started. In other words, the fact that an incremental preverbal message is completed can only be established in retrospect. The reason for this is that preverbal messages (semantic structures) may be extended incrementally ad infinitum: ‘In an incremental mode of sentence generation […] a sentence which is in principle complete may often be extended by adding another modifier or even a case relation.’ (De Smedt 1990a: 83) This can be done by three mechanisms in incremental grammatical encoding: expansion, coordination, and correction (De Smedt 1990a: 16). The latter two were already discussed in connection with underspecification formalisms in section 5.3. The first one is the normal mode of operation when generating a preverbal message incrementally.

Incremental preverbal messages as they are produced by pvm-generation can be rewritten so that they conform to semantic representations as they are proposed by Jackendoft, cf. Guhe (in print). The basic idea is that the increments belonging to one incremental preverbal message are collected until the preverbal message is complete. Then this representation is rewritten so that it is equivalent to a representation that was not generated in an incremental fashion but en bloc. One can imagine this as having two different views on the same representation, a temporal and a non-temporal view. In the temporal (incremental) view the emphasis lies on the succession of increments, i.e., the generation of the representation is emphasised. In the non-temporal view, which is traditionally taken in linguistics, all increments belonging to an incremental preverbal message are considered simultaneously. While in the first view the main interest is in the processing mechanisms, in the second view it is in the information that is required for an adequate semantic representation.

Both views can profit from each other. One result of considering the processing mechanisms is that the focus and/or perspective of a semantic representation can be encoded by the order of the components of the representation. Considering the non-temporal properties of a semantic representation makes it possible to concentrate on its informational content so as to formulate the goal representation of the generation process. A model like the one in section 5.3 can then in the end be tested whether the generated preverbal messages are equivalent to these semantic representations. (However, the model is not yet detailed enough to perform such elaborated tests.) In sections 5.6 and 5.4 proposed as a working hypothesis that the first element is the item that is in the focus of the utterance. Additionally, each time an element is chosen...
for verbalisation the perspective of the utterance is narrowed down.

\textit{pvm}-generation must observe two crucial conditions, cf section 2.3. Firstly, for each increment of an incremental preverbal message it must be ascertained that it fits to the increments already generated. This is the \textit{consistency condition}. Secondly, all increments of an incremental preverbal message taken together must form a complete semantic representation as it is required by accounts like the one by Jackendoff (1990). This is the \textit{completeness condition}. Meeting these conditions is a result of the functioning of inC. In particular there are no explicit computations whether this is the case (consistency and completeness checks). This is a further point in favour of the cognitive adequacy of inC: humans do not perform explicit reasoning about whether a sentence is complete. Knowledge about this is implicit. The only way to detect an inconsistency or the incompleteness of an utterance in the Levelt model is via self-monitoring.

There is an important difference between a sequence of increments like an incremental preverbal message and a dynamic representation like the \textit{CCR}. A sequence of increments is a richer representation in that it contains the information about which increment was generated at which point in time. This information is not stored in a representation like the \textit{CCR}. Furthermore, in contrast to the information stored in the \textit{CCR}, which simply exists as long as \textit{inC} is running – remember that there is no ‘forgetting mechanism’ –, incremental preverbal message have a twofold character. Seen as succession of increments between conceptualiser and formulator they are non-persistent representations, whereas in the traverse, which is part of the \textit{CCR}, they are permanent representations as well.

\textit{pvm}-generation uses a particular notion of preverbal messages, the \textit{current preverbal message}:

\begin{term}
Current preverbal message. The \textit{current preverbal message} is the preverbal message that is currently produced by \textit{pvm}-generation. The increments contained in the current preverbal message are accessible by \textit{pvm}-generation so that it can (a) keep track of still missing parts of the incremental preverbal message and (b) compute extensions, modifications, and self-corrections.
\end{term}

Current preverbal messages are used to keep track of what has already been said. This is important, for example, when \textit{pvm}-generation decides which designations to use for verbalising a refO, cf the next section. Additionally, it is the means to detect the need for a self-correction in the case of conceptual changes. These are changes to elements in the conceptual representation that are part of the current preverbal message. In other words, if \textit{pvm}-generation detects that a refO that is used in the current preverbal message changed after it was sent to the formulator,

* This is true at least under a certain perspective. Such a representation can also be seen as a succession of states of the representation. However, at a given point in time the order in which the elements of the representation were inserted or modified plays no role and is not preserved. In structures like an incremental preverbal message this is different.
it can initiate a self-correction. For instance, if the PPU computes that the plane of
the introductory example does not move towards GATE-B21 but towards GATE-B23
the following utterance could result, cf also example (3) on page 102:

(1) Flug CK-314 bewegt sich auf Gate B21 zu ... äh nein ... auf Gate B23.
Flight CK-314 moves towards gate B21 ... uh no ... gate B23.

In the algorithms given in the following the detection of conceptual changes will
not be discussed further for reasons of brevity.

13.2 Incremental generation of preverbal messages

Before I start with the description of how incremental preverbal messages are gen-
erated by PVM-generation I want to make some introductory remarks. According
to De Smedt (1990a) the following information is required by the formulator:

1. semantic concepts (that refer to entities, events)
2. case relations (‘deep’ case, i.e. semantic roles between concepts)
3. features (definiteness, number, etc)

The algorithm presented in the following only generates the first two kinds of in-
formation. Features required by a formulator must be added to PVM-generation
when nC is actually connected to one, because the question of which features are
required strongly depends on how language specific the preverbal message will be.
For example, it is quite likely that a formulator generating in a language that not
only makes a singular/plural distinction but also knows a dual needs a threefold
distinction of number in the preverbal message while in other languages a twofold
distinction suffices. In fact, using a threefold distinction may even be disadvant-
ageous in the latter case, because it adds unnecessary information, which would
increase run-time.

Similarly, the problem of encodability (also called the generation gap problem,
Meteer 1990) is not considered in the following. This problem addresses the fact
that the conceptualiser must ascertain that the formulator is able to encode a pre-
verbal message linguistically. This is a hard problem when the full productivity
and complexity of a natural language shall be exploited. (It can rather easily be
achieved when only a small subset of the natural language is used.) A conceptual-
iser may, therefore, strongly profit from a formulator that uses ‘productive lexical
rules which derive categories from other ones, e.g. nominalization and passiviza-
tion.’ (De Smedt 1990a: 148f)

The algorithm presented below bears some resemblance to the so-called incre-

* This is only a single algorithm, not a class of algorithms as I defined it in part II.
The Dale & Reiter algorithm is the first one of a whole series of algorithms for the generation of referring expressions. Thus, the algorithm of PVM-generation is much more general. One of the main advantages of this algorithm for the purposes at hand is that it possesses some psychological plausibility, because its results are in accordance with empirical findings, e.g., the ones by Pechmann (1984) and because it implicitly observes the Gricean maxims (Grice, 1975). Some extensions are proposed in van Deemter (2002). Krahmer & Theune (2002) describe a variant for the generation of reduced expressions in a discourse context. For example, the tiny, red, cute dog may be referred to simply as the dog when it is mentioned for a second time. A multimodal variant is given in van der Sluis & Krahmer (2001), where the generation of the referring expression is supported by a pointing gesture of variable precision. Some extensions are proposed in van Deemter (2002). Krahmer & Theune (2002) describe a variant for the generation of reduced expressions in a discourse context. For example, the tiny, red, cute dog may be referred to simply as the dog when it is mentioned for a second time. A multimodal variant is given in van der Sluis & Krahmer (2001), where the generation of the referring expression is supported by a pointing gesture of variable precision. A proposal of how this algorithm may be cast into graph-theoretical terms is given by Krahmer, van Erk, & Verleg (2003). The latter allows to use results from the study of graphs for improving the algorithm. The Dale and Reiter algorithm has more or less become the baseline algorithm for this task in the NLG field. A version of the algorithm can be used at the point where referring expressions are generated, which is not done up to now.

The fact that PVM-generation does not perform a linearisation of any kind is in agreement with assumptions of models of incrementally working formulators. Such formulators, e.g., IF by De Smedt (1990a,b) or Performance Grammar (Kempen, forthcoming; Kempen & Harbusch, 2002, in print), explicitly acknowledge that input increments do not come in a pre-specified order. The latter uses a slot–filler model for the positioning of increments (phrases/segments), i.e., the generation of an element is delayed if some grammatically required element is still missing.

The last precursory remark is that in what follows one general principle should be kept in mind. Deciding upon a verbalisation is always a two-step process in INC. First, it decides to verbalise a refO and only then how to verbalise it. This is true for the division of labour between selection and PVM-generation as well as for PVM-generation's algorithm. Since refOs usually contain lots of designations, some of which refer to other refOs, the designations that shall be used in the verbalisation have to be picked from a refO in the second step. Following this, the referenced refOs with further designations (with further refOs and so on) are verbalised. The large number of designations makes constraints for choosing them indispensable, because not all information provided by the designations is needed in a verbalisation.

The PVM-generation process consists of three main parts. Apart from the outer loop that has the same function as the ones in the other processes, there is the procedure Verbalise and the function SelectDesignations, which are given in figures 13.1, 13.2, and 13.3, respectively. The outer loop simply watches the head of a traverse buffer (head_of_TB) and waits that the defined latency (parameter LT) expires. In this case it fetches the head of traverse buffer and starts a new current preverbal message with the refO the head of traverse buffer is pointing to as first refO. (For reasons of brevity I will, again, in the following not distinguish the pointers in the traverse buffer from the actual refOs in the ccr the pointers refer to.) Finally, the procedure Verbalise is called, which generates the verbalisation for
The head of traverse buffer. This loop runs until the linearisation process terminates and pvm-generation can synchronise with the monitor process that nothing more is to be done, cf chapter 15. This synchronisation is necessary, because the monitor process interacts directly with pvm-generation when it detects the need for a self-correction.∗

From this way of operation follows that an incremental preverbal message (a verbalisation) must be finished before the next head of traverse buffer can be taken. This means two things. Firstly, it is not ascertained that the head of traverse buffer is verbalised immediately after the latency expired. What is more, this means that the head of traverse buffer can be removed from the traverse buffer by an operation of the selection process without being verbalised even after its latency expired. Secondly, the generation of an incremental preverbal message cannot be interrupted, ie selection cannot decide not to verbalise a situation after the generation of the corresponding preverbal message commenced. After initiation of an incremental preverbal message the 'normal' operation of pvm-generation is only interrupted when the monitor detects an error and inserts a self-correction or when pvm-generation detects a conceptual change and makes the corresponding self-correction or extension, cf section 5.5. In order not to complicate the description of the functioning of pvm-generation both possibilities are not discussed here.

The procedure Verbalise, cf figure 13.2 generates a verbalisation for a refO. When it is called for the first time it receives the head of traverse buffer as argument. Since the verbalisation of a head of traverse buffer involves the verbalisation of further refOs, Verbalise calls itself recursively. The first operation Verbalise carries out is to append the refO it is currently verbalising to the current preverbal message (current_pvm).† Then, a new verbalisation refO is generated. After its

---

∗ As I already stated the monitor process is only implemented very sketchy, because formulator and language comprehension system are missing. For this reason the exact termination conditions are not discussed here.

† The current preverbal message does not contain verbalisation refOs but pointers to the refOs in the ccr, which is the prerequisite for detecting conceptual changes. (Conceptual changes are detected by comparing the refOs in the ccr to which the pointers in the current preverbal message refer to the
PROC Verbalise(refO)
  var verbalisation_refO, #the newly created verbalisation refO
       selected_desigs #designations selected for verbalisation of the refO
  Append(refO, current_pvm)
  verbalisation_refO ← new refO
  selected_desigs ← SelectDesignations(refO)
  AddDesigs(selected_desigs, verbalisation_refO)
  Append(selected_desigs, used_desigs)
  Append(verbalisation_refO, output_stream)
  Append(verbalisation_refO, traverse)
  FOR ALL r ← RelatedRefOs(selected_desigs) DO
    Verbalise(r)
  ENDFOR
ENDPROC

Figure 13.2: The procedure Verbalise

creation it only contains the attributes of the original refO and the information
of which refO it is a verbalisation. Of the designations only those actually used in the
verbalisation are added to the verbalisation refO.

The selection of designations is performed by SelectDesignations. The result
is stored in selected_desigs so that subsequent calls of SelectDesignations
do not select these designations again. (Since the list selected_desigs is defined
in Verbalise it is also visible within SelectDesignations.) The selected designa-
tions are added to the verbalisation refO. If a selected designation is a description
it is stored in its resolved form, eg if \( \eta x \text{ chpos}(x, r_2, r_3) \) is a description of \( r_1 \) it
is stored as \( \text{chpos}(r_1, r_2, r_3) \). This way it will not be selected if it is considered
again when designations for refOs \( r_2 \) and \( r_3 \) are selected, see below. With this, the
verbalisation refO is complete. It is sent to the formulator as next increment of the
incremental preverbal message and appended to the traverse. Finally, Verbalise
calls itself for the refOs that the verbalised designations refer to.

Three kinds of constraints are used in the selection of designations, structural,
activation, and conceptual. Of these, the structural constraints are most important;
they form the ‘backbone’ of the algorithm of SelectDesignations. In section 14.2
I will discuss a worked example that demonstrates the interplay of these three con-
straints in detail. Let me now first illustrate them in more detail before describing
SelectDesignations.

The structural constraint says that only grounded designations are verbalised. If
designations refer to no other refO they are directly grounded, eg names or the
description \( \eta x \text{ plane}(x) \). Designations pointing to other refOs are, consequently, not
directly grounded, eg \( \eta x \text{ chpos}(x, r_2, r_3) \). In order to verbalise them it must be
tested whether they are groundable. In the chain of grounding, all refOs that the

197
designation contains are checked whether they have a grounded designation. In the example this must be done for \( r_2 \) and \( r_3 \). If a designation is groundable by a chain of grounding the designation is *indirectly grounded*. A designation evaluated in the chain of grounding is *cyclic* if it has already been used in this chain, and the grounding attempt fails. An example of a cyclic designation is as follows. Assume that for the verbalisation of \( r_1 \) the description \( \eta x \ \text{chpos}(x, r_2, r_3) \) has been chosen and is part of the current preverbal message. This also means that it was appended to the selected_desigs list in its resolved form, \( \text{chpos}(r_1, r_2, r_3) \). In the next step a designation for \( r_2 \) must be found. One possibility would be to select \( \eta x \ \text{chpos}(r_1, x, r_3) \). However, choosing this description would add no new information to the current preverbal message, because it can also be reduced to \( \text{chpos}(r_1, r_2, r_3) \). For this reason it is a cyclic designation that is not selected. This also holds for \( r_3 \) and \( \eta x \ \text{chpos}(r_1, r_2, x) \).

The *activation constraint* evaluates the activation values assigned to the designations in question. It checks whether the activation of a designation is above the activation threshold (parameter \( \alpha T \)). However, if a designation is necessary for a semantically complete preverbal message it may be chosen despite an activation value below \( \alpha T \). In the motion events investigated there is one case in which this constraint can be violated, viz selecting a designation for a situation \( \text{refO} \), where it is not considered at all. In the other cases this is not necessary, because there is also the possibility to generate a verbalisation \( \text{refO} \) that contains no designations. Such \( \text{refOs} \) are generated if, for example, they have been verbalised before, ie if they were used in previous preverbal messages, cf also section 13.3.

The *conceptual constraints* evaluate whether a designation is conceptually consistent with the designations chosen so far. Up to now, there is only one conceptual constraint, the *homogeneous-part-of constraint*. If a \( \text{refO} \) is tested in the chain of grounding that is part of another \( \text{refO} \) in the current preverbal message and the \( \text{refOs} \) in question are of a *homogeneous sort*, no designation from the \( \text{refO} \) will be used in the verbalisation. An entity is of a homogeneous sort if its parts are of the same sort as the entity. An example is a path, because parts of paths are paths.*

The function SelectDesignations evaluates these constraints, cf figure 13.3. It first copies all designations of the \( \text{refO} \) under consideration to a list† and collects the non-cyclic designations of the \( \text{refO} \). In the next step the function distinguishes between the verbalisation of a \( \text{refO} \) having sort situation and a verbalisation of a \( \text{refO} \) having another sort. In the former case the designation must be non-cyclic, and it must have a predicate that can be used for describing a situation. In particular a designation must describe what happens in this situation. situation \( \text{refOs as}

---

* I regard paths as extended entities, and points lying on a path are not formalised as *parts* of a path but as *coinciding* with a path [Eschenbach, Habel, & Kulik 1999, Eschenbach, Tschander, Habel, & Kulik 2000]. Thus, a point, eg a location, cannot be part of a path, but a path can have multiple sub-paths.

† Apart from the fact that this enhances performance it also ascertains that the function does not operate on a changing \( \text{refO} \). This means, for instance, that if a designation is added while SelectDesignations is running it is not considered.
13.2 INCREMENTAL GENERATION OF PREVERBAL MESSAGES

fun SelectDesignations(refO)

var
designs, #designations of refO
dg_desigs, #directly grounded designations
non_cyclic_designs, #designations without cyclic ones
selected_designs, #designations selected for verbalisation; return value
sit_desig_preds, #designation predicates for a situation refO

designs ← Designations(refO)
non_cyclic_designs ← AllNotCyclic(desigs, refO)

if Sort(refO) = situation then
selected_designs ← Filter(non_cyclic_designs, sit_desig_preds)
else
dg_designs ← Filter(non_cyclic_designs, DirectlyGrounded)
for all d ∈ dg_designs do
if Activation(d) and ConcConsistent(d) and d /∈ used_designs
then
Append(d, selected_designs)
endif
endfor
Append(Filter((non_cyclic_designs \ dg_designs \ used_designs),
(Groundable and Activation and ConcConsistent),
selected_designs)
endif
return selected_designs
endfun

Figure 13.3: The function SelectDesignations

they are used here contain only few designations. For this reason a simple Filter operation suffices to find an appropriate one; the constraints need not be evaluated further, at least not until the examples investigated become more complex.

If the refO to be verbalised has another sort than situation the directly grounded designations (dg_designs) are filtered from the non-cyclic designations of the refO. All of these designations must also fulfill the activation constraint and the conceptual constraints. Additionally, they must not be used in a previous preverbal message, i.e., they must not be in the used_designs list. This list is stored at the top-most level of PVM-generation, cf figure [3.1]. Thus, a designation can be used only once as long as INC is running. The selected designations are appended to the used_designs list in the function Verbalise (figure [3.2]). The designations that

situation refOs are not tested for this condition. Otherwise a situation could not be verbalised a second time, which is, for example, necessary when it is verbalised as an expected situation and then again after the situation actually took place. Allowing a designation to be verbalised only once is a strong oversimplification, which must be modelled differently in the next INC versions. This is an assumption to 'get the implementation running'.

199
meet these conditions are selected for verbalisation, i.e. they are appended to the selected_desigs list.

After this, the remaining designations, i.e. the non-cyclic, not directly grounded designations, are evaluated whether they are groundable, conform to the activation and conceptual constraints, and whether they were not already used in a verbalisation before. The designations fulfilling all these conditions are also appended to the selected_desigs list. The list containing all of the designations selected this way is the result of the function SelectDesignations.

13.3 Sequences of incremental preverbal messages

In the previous section I described how incremental preverbal messages are produced by pvm-generation. This is the current state of the implementation. However, a verbal description of a scene like the one in chapter requires a sequence of multiple preverbal messages – apart from A plane is docking. On that basis of what I described so far the following two questions come to mind:

1. How are preverbal messages related to each other?
2. How are the parts of preverbal messages related?

In the context of online descriptions of events a first answer to the first question is that the temporal relations between the situations described in the preverbal messages must be part of the discourse structure. In other words, the temporal relations that are represented as refO attributes in the ccr can be used to interrelate preverbal messages. Referential nets themselves offer a first answer to the second questions, because they facilitate the generation of co-references between increments of preverbal messages. Since the increments of preverbal messages are refOs, the verbalisation refOs referring to the same ccr refO are co-referential.

In this section I will present a proposal of how the current version of inc can be extended in order to accomplish this. It mainly affects the pvm-generation process. The proposal is published in more detail in (Guhe, Habel, & Tschander) in print.

Consider the scene shown in figure. The spatial background of the scene consists of another part of the manoeuvering area already used in the example in chapter. It contains two parallel runways (runway-1 and runway-2) and a way connecting them (connection). In this scene two planes are moving. Their movements become observable at different points of time, indicated in the figure by the time-stamps t1 and t3. The time-stamps depicted in figure are numbered according to their temporal ordering (t1 < t2 < t3 < t4). Furthermore, they correlate with the positions of the planes during their movements that subdivide the scene into different situations. In extension of the segmentation points used so far – starting and endpoints of movements – the ppu identifies the segmentation points for this scene where
1. a plane appears in the scene,
2. the direction of a movement changes,
3. a movement ends.

In this scene a plane appears on Runway-2 at $t_1$ and moves straight until it reaches the Connection way between Runway-1 and Runway-2, where it turns off at $t_2$. Then, at $t_3$ a second plane appears on Runway-1 and moves straight ahead. At $t_4$, finally, the first plane stops. A verbalisation of this scene is given in (2).

(2) a. *Ein Flugzeug fährt auf Runway 2.*
   'A plane is moving on Runway 2.'

b. *Nachdem es auf die Verbindung abgebogen ist,*
   'After it has turned off onto the Connection,'

c. *fährt ein anderes Flugzeug auf Runway 1.*
   'another plane moves on Runway 1.'

d. *Dann stoppt das erste Flugzeug.*
   'Then the first plane stops,'

e. *während das Flugzeug auf Runway 1 weiterfährt.*
   'while the plane on Runway 1 is moving on.'

This example is made up, because participants usually produce verbalisations that are much 'noisier', ie, for example, they often produce incomplete sentences or make hesitations. Some verbalisations for scenes very similar to the one discussed here are given in appendix C. They show that the means of verbalisations discussed below are indeed used, if not in such a concise and 'literate' form. However, one as-
pect of the verbalisations is not captured by INC, namely that the participants make a lot of statements about the causal relations within the scene. The most prominent point is that they believe that the first plane stops in order to let the second plane pass. A possibility to infer this is to extrapolate the movements of both planes and see that they will crash into each other if they continue their movements unchanged. So, in order that this does not happen, one plane has to stop. Reasoning of this kind, especially reasoning about future states that will not or should not occur is beyond INC’s inferential capabilities. INC builds up complex representations for the only reason that there is a corresponding rule in the CS.

I will not go into the details of the conceptual representation, but see figure 13.5 for the event structure. The events with index $p_1$ are the movements of the first plane (the one moving on runway-2 and the connection), the events with index $p_2$ the movements of the second plane. Thus, in this scene there are two event threads (term 12.1 on page 189) that partition the event structure of the conceptual representation.

There is a new type of situation, viz turn off, the punctual event when the plane changes its direction and turns off onto the connection. Regarding turning off as punctual event requires a slight abstraction, because in fact it consists of the point where the path initially changes its curvature, the movement along the curved path, and the point where the movement becomes straight again. Such an analysis is used in the example in section 14.2, but I will not elaborate on this sub-structure here. Firstly, although it would be more accurate, it would also add a lot of details. Secondly, in the verbalisations given in appendix C the participants indeed use the abstracted turn off, which can be seen by the frequent use of the verb abbiegen ‘turn off’.

In order to generate a verbalisation like INC as I described it up to now must be extended in three respects:

1. construction must keep track of the temporal relations between the on-going events in the scene,
2. PVM-generation must decide how the temporal relations are verbalised, and
3. PVM-generation must evaluate how the co-references are expressed.

Extending construction is necessary, because it must establish the temporal relations between the currently on-going events of different event threads. Otherwise,
the utterances in (2b)–(2c) and (2d)–(2e) cannot be generated so that they explicitly mention the temporal relations between the events, which are expressed by the temporal connectives nach dem 'after', dann 'then', and während 'while', 'during'. The temporal relations can be expressed by a constraint net. For the example the end state of the net is given in figure 13.6. The grey relations are continuations of the scene that were once possible but did not occur. Remember that the relations differ with respect to whether they hold between extended situations, or punctual situations, or punctual and extended ones, cf section 4.5.

When a temporal relation is generated by pvm-generation, two cases must be distinguished:

1. the temporal relation connects the preverbal message with a previously generated one, and
2. the temporal relation connects this and a following preverbal message.

For the first case the current mode of operation must only be extended by a computation that determines the temporal relation of the situation that is about to be described by the preverbal message with a previously generated one. This step is necessary, if, for example, there is more than one possible relation. Assume that the incremental preverbal message for utterance (2a), which describes CHPOSp1, starts with the verbalisation refO v1. The skeleton of the corresponding refO sequence is given in 3.1.
The next preverbal message for utterance (2b) – verbalising TURN_OFFP1 – is now connected to the previous one by discourse refO v4. It starts with v5, and the temporal relation, which can be encoded by the formulator as a temporal connective, is given as functional expression finished_by(v1, v5) at the discourse refO. Since there is no way to connect whole incremental preverbal messages, because there is no refO for the complete preverbal message, the temporal relation is given between the first increments. Since only situation refOs are present in the traverse buffer, these are the only refOs that can have this function.

However, this is only the first step in generating the preverbal messages for (2b)–(2c), because the temporal relation between both must be known in advance. Otherwise, nachdem ‘after’ cannot be generated at the beginning of the utterance. While the other mechanisms discussed in this section only extend INC, this problem requires a real change, because pvm-generation needs access to more elements of the traverse buffer than just the head of traverse buffer. In order to obtain an utterance like (2b)–(2c) the traverse buffer must contain pointers to the situations TURN_OFFP1 and CHPOS_P2, and both must be verbalised together. Example (4) shows how the three incremental preverbal messages are generated in this case.

* (2d)–(2e) can be generated without this, because dann ‘then’ connects (2d) to the previous utterance.
As in the simpler case (5) given above, \( v_4 \) connects the forthcoming preverbal message with the previous one, while \( v_5 \) connects the next two. Both must be planned together, because the temporal relation \( \text{before}(v_6, v_7) \), which can result in the temporal connective \textit{nachdem} ‘after’, must be available before the first incremental preverbal message is started. A consequence of this generation method is that \( v_7 \) is announced rather early and several other verbalisation refOs are generated before it is actually generated. With regard to the distinction of linear and hierarchical incrementality, cf page \( 88 \), this means that not only generating an incremental preverbal message involves hierarchical incrementality but that the sequences of incremental preverbal messages add another level to the hierarchy.

The third item in the list on page 202 is to interrelate parts of incremental preverbal messages by co-references, which can result in anaphora or ellipses. As I already pointed out, the referential nets formalism makes this task rather easy. Since co-references are mostly important for generating referring expressions, an approach similar to the algorithm of [Dale & Reiter 1995] extended by [Krahmer & Theune 2002] can be used. Three different verbalisation refOs can be generated from the nx plane\((x)\) description of \( r_3 \), given as \( v_1 \) to \( v_3 \). The verbalisation refOs given in the following will only contain the information that is important for the discussed examples, although the verbalisation refOs actually generated may require more information.

(5)  plane \( r_3 \)  

\( \text{nx plane}(x) \)
\( \text{nx chpos(chpos}_{P1}, x, \ldots) \)
\( \text{nx chpos(TURN}_{OFF}_{P1}, x, \ldots) \)

(6) a.  plane \( v_1 \)  

\( \text{verb}_\text{of}(r_3) \)  

\( \text{nx plane}(x) \)

b.  plane \( v_2 \)  

\( \text{verb}_\text{of}(r_3) \)  

\( \text{ix plane}(x) \)

c.  plane \( v_3 \)  

\( \text{verb}_\text{of}(r_3) \)

\( v_1 \) corresponds to a verbalisation of the referring expression \textit{ein Flugzeug} ‘a plane’; \( v_2 \) to \textit{das Flugzeug} ‘the plane’. The co-reference of both verbalisation refOs is expressed by \( \text{verb}_\text{of}(r_3) \), i.e. the ccr refO from which both verbalisation refOs were generated is \( r_3 \). Thus, both verbalisations refer to the same entity. \( v_3 \) shows the case
that no designation was chosen, which currently occurs in \(\text{INC}\) if the designation was already used before, cf section 13.2. Thus, in (2a) a verbalisation refO like \(v_3\) is used, because the \(\eta_x \text{plane}(x)\) description, from which the alternative \(\iota_x \text{plane}(x)\) would be computed, was already used in (2a).

If there is more than just one possible referent of a (reduced) referring expression, eg both planes are possible referents in (2c)–(2e), it is sometimes necessary to suppress co-references. Linguistic means to do this are *ein anderes Flugzeug* ‘another plane’ (\(v_4\); assume that the second plane is represented by \(r_{20}\)), *das andere Flugzeug* ‘the other plane’ (\(v_5\)), and *das erste/zweite Flugzeug* ‘the first/second plane’ (\(v_6\)). The indefinite variant is chosen in the first mentioning, the definite ones afterwards.

(7) a.  
\[
\begin{align*}
\text{plane} & \quad v_4 & \quad \eta_x \text{plane}(x) \\
\text{verb}_{\text{of}(r_{20})} & & \eta_x \text{different}(x, v_1)
\end{align*}
\]

b.  
\[
\begin{align*}
\text{plane} & \quad v_5 & \quad \iota_x \text{plane}(x) \\
\text{verb}_{\text{of}(r_{20})} & & \iota_x \text{different}(x, v_1)
\end{align*}
\]

c.  
\[
\begin{align*}
\text{plane} & \quad v_6 & \quad \iota_x \text{plane}(x) \\
\text{verb}_{\text{of}(r_3)} & & \iota_x \text{first}(x, v_1)
\end{align*}
\]

Another possibility for referring to an entity is used in (2e), where the plane is identified by its position in the world, *das Flugzeug auf Runway 1* ‘the plane on Runway 1’. This is accomplished by generating additional verbalisation refOs from the spatial structure given in the \(\text{CCR}\) that locate the referent (the plane).

With these extensions \(\text{PVM}\)-generation can already generate rather realistic online descriptions of events. However, since it still has many possibilities for generating utterances describing a scene, \(\text{INC}\) would probably profit from integrating a more elaborate activation model, which would indicate which elements of the \(\text{CCR}\) are prominent and, therefore, prone for verbalisation.
SIMULATIONS

Simulations are the means to evaluate the cognitive adequacy of \textsc{iNC}. While up to now the description of \textsc{iNC} concentrated on its architecture and algorithms, in this chapter I will present its implementation and how this is used to carry out simulations. Since investigating conceptualisation suffers from the difficulty of the great variability of the data recorded in verbalisation studies, the approach taken with \textsc{iNC} is to account for this variability on the one hand, while on the other hand also determine the default mode of operation, in which the most common verbalisations can be reproduced. This is done by different settings of the parameters described in section \ref{sec:parameters}. The default behaviour is determined by finding default values for these parameters.

The results presented in this chapter were already published previously. The material discussed in section \ref{sec:example1} in Guhe, Habel, & Tschander (2003a,b); the material of section \ref{sec:example2} in Guhe & Habel (2001).

14.1 The \textsc{iNC} simulator

\textsc{iNC} is implemented in the Mozart /Oz programming system.\footnote{Information about Mozart /Oz can be obtained from \url{http://www.mozart-oz.org}.} \textsc{iNC} proper is a Mozart /Oz module programmed for interaction with other modules. The \textsc{iNC} module can be called by other modules (programs). For performing the simulations there is an interface, which is simply called the \textsc{iNC} simulator.\footnote{The current version of the \textsc{iNC} implementation is together with the simulator can be downloaded from \url{http://www.informatik.uni-hamburg.de/WSV/sprachproduktion/software/inC-en.html}.} The simulator is shown in figure \ref{fig:simulator} just before it executes a simulation for the example discussed in the next section.

In the top left of the simulator window the four parameters for a simulation are specified. Next to this the domain is chosen. Apart from the two domains discussed in this investigation – motion events and the generation of sketch maps – it is also possible to specify a user-defined domain. The main effect using different domains is that different concept storages are used.\footnote{Thus, specifying an own domain means to supply a file containing the \texttt{cs}.} The logging output can be sent directly
to a window, a file, or both. The rightmost field serves to let InC run in a simulated real-time condition. This is due to the current limitation that the PPU is not capable to connect directly to InC when it generates its output. Instead, it writes its output, which contains a sequence of pes, to a file. InC needs to synchronise to the output of the PPU, because it does not generate output at a constant rate, i.e., the time intervals between the produced pes vary. If no real-time behaviour is required then this option can be switched off, which only marginally influences the behaviour of InC, as the simulations showed. The field below the checkbutton specifies the time of one frame of the PPU, usually 50 ms. Below these fields the selection strategy can be chosen. Most Complex stands for the standard selection strategy shown in figure 11.2 Retain Granularity for the alternative one, given in figure 11.3.

The bottom fields serve to specify the files that are used by InC. The PPU file is the output of the PPU (thus, the input to InC), the PVM file is a text file containing the incremental preverbal messages (thus, the output of InC), and the log file is a file generated while InC is running, containing a log of InC’s decisions and results. The settings file, finally, is used to store all the values just described.

The left of the two buttons at the bottom of the simulator window opens another window in which the settings for the graph-drawing tool are specified with which the referential nets of the ccr and the cs are visualised. This tool is called

* Since InC is indeterministic, this is a valuable means for reproducing the actual sequence in which InC performed its computations.
DaVinci. Among other things it is possible to filter out certain kinds of refOs in this window, say all verbalisation refOs, because even simple referential nets are quite complex when visualised as graph, which is mainly due to the many interrelations by attributes and designations. DaVinci is also able to draw the referential nets incrementally, i.e. adding nodes and edges as they become available. A graph drawn by DaVinci for the example discussed in the next section is given in section B.4. The button in the lower right corner starts the INC simulation.

14.2 A worked example

The example simulation I will present in this section is taken from Guhe, Habel, & Tschander (2003a,b). I use this example instead of the one from chapter 1 because it is simpler and suffices for the purpose at hand — and is complex nevertheless. It consists of a plane taxiing straight forward and then, starting at the position indicated by the dot, moving to the left, cf figure 14.2. (The participants in the corresponding verbalisation study do see neither the dot nor the dashed line.) The plane moves before a white background. The continuous movement is subdivided at the position of the dot; there is one sub-situation where the plane moves forwards and one where it moves to the left. The final state of the scene is given in figure 14.3. There are many possible verbalisations of this motion event. Some are given in (1) to (3). Utterance (1) describes the whole scene without going into the details of the event structure, while (2) and (3) take into account the sub-events. The latter two verbalisations also express the temporal order of the sub-events.

(1) *Ein Flugzeug fährt nach links.*
‘A plane moves to the left.’

The incremental preverbal messages described in this section were produced without the extensions described in section 13.3. Thus, there are no explicit computations regarding temporal relations and co-references.
Figure 14.3: Final state of the ccr for the plane turning left
(2) *Ein Flugzeug bewegt sich geradeaus und fährt eine Linkskurve.*
'A plane moves straight on and follows a left curve.'

(3) *Ein Flugzeug fährt und biegt nach links ab.*
'A plane moves and turns off to the left.'

The output that is actually produced by the system for this example when using the default values for the parameters is given in appendix B. Since this output consists of text-files, which are less readable than the drawn referential nets used so far, I will continue to use this mode of presentation for now.

**Parameters.** The default values for \( \text{INC} \)'s parameters are:

- \( \text{DOAT} = 0.5 \)
- \( \text{LOTB} = 6 \)
- \( \text{LT} = 125 \text{ ms} \)
- \( \text{AT} = 0.5 \)

With these values \( \text{INC} \) exhibits ‘normal’ behaviour. The value for \( \text{DOAT} \) ascertains that expectations are only generated if already a substantial amount has been perceived. The value for \( \text{LOTB} \) makes sure that nothing is lost due to the size limitation of the traverse buffer. This usually only happens with values of 1 or 2. An \( \text{LT} \) of 125 ms has the effect that the situations that were selected by the selection process are indeed verbalised. The only exception is the case in which a new \( \text{PE} \) is read in and a complex concept containing this new \( \text{PE} \) is created. If selection uses the standard selection strategy the simpler element of the traverse buffer is replaced by the more complex one. Finally, an \( \text{AT} \) of 0.5 means that only rather prominent designations are chosen for verbalisation.

**Construction.** The end state of the \( \text{CCR} \) for the example that is built up by construction is shown in figure 14.3. There are two points in time when construction obtains information about new \( \text{PEs} \); all other information sent by the \( \text{PPU} \) updates already existing \( \text{PEs} \). The first point is when the plane becomes visible. Then the \( \text{PPU} \) sends information about three \( \text{PEs} \), which result in refOs \( r_1 \sim r_6 \). The second is when the plane starts moving to the left, which brings about \( r_7, r_8, \) and \( r_{10} \). \( r_9 \) and \( r_{11} \) are inserted due to results of the concept matcher. Since the triple of object, situation, and \( \text{spatial}_{-}\text{entity} \) was already repeatedly mentioned, I will not describe it further this time.

Each path induces a spatial reference system, which I did not elaborate on in chapter 16. It is introduced by three additional \( \text{PEs} \) coming from the \( \text{PPU} \). The first one introduces \( r_4 \) of sort \( \text{ref}_{-}\text{sys}_{-}\text{spat} \). It is connected to \( r_3 \) – the path generating the reference system – via a functional relation. It is a functional relation, because \( r_4 \) is the only spatial reference system of \( r_3 \). At the same time the path subdivides

Since the implementation can currently not deal with functional expressions I used a i-abstracted description instead, which is only a minor difference in this example.
the space into two half-planes with respect to its reference system, the left region and the right region. These are the other two pes. r5, the left region of r4, is related to r4 via the functional relation Left_region; the right region (r6) via Right_region.

When the plane starts moving to the left the ppu sends the second set of new pes. The new pes result in the creation of the path refO r7, which represents the curved part of the movement, the location refO r8, where the two paths meet, and the corresponding situation refO r10. According to the rule given in figure 10.4 the two path refOs are grouped to r9 for the overall path and the two situations to r11 for the overall situation.

Selection. Since the event structure of this example consists of only three situation refOs (r2, r10, r11), the selection process can select only two different event sequences when the parameters are set to the default values. If the standard selection strategy is used with the default values for the parameters the first situation refO (r2) and the situation refO encompassing the whole movement (r11) are selected. If the alternative selection strategy is used, which retains the level of granularity as far as possible, the two simple movements (r2 and r10) are selected.

pvm-generation. To simplify matters in the following, let us assume that the scene has been completely perceived. Let us further assume the parameters are set in a way that only complex situation r11 is verbalised. The required setting is LT \( \geq 6050 \) ms, because r11 must be generated by construction before the first simple situation representing the movement along the straight path (r2) has been taken out of the traverse buffer by pvm-generation. (Only for LT = 0 ms all situations are verbalised.) The parameter values for the simulation I describe in the following are doat = 0.5, loth = 6, LT = 6050 ms, at = 0.5. See appendix B for the actual output generated by Inc. I chose the verbalisation of r11 as example for illustrating the functioning pvm-generation, because it requires the most complex computations by this process for this scene.

pvm-generation starts the generation of the incremental preverbal message for r11 with choosing description \( \eta x \) chpos(\( x, r1, r9 \)). Since no other designations have been generated so far it cannot be cyclic. Hence, it is chosen, sent to the formulator, and appended to the traverse.

Then, it has to be determined how the two refOs referred to can be described. r1 contains four descriptions. \( \eta x \) chpos(\( r11, x, r9 \)) is cyclic. Like the first description it can be reduced to chpos(\( r11, r1, r9 \)) and is, therefore, not considered further. \( \eta x \) plane(\( x \)) is directly grounded and conforms to the other constraints, for which reason it is chosen. When trying to ground \( \eta x \) chpos(\( r2, x, r3 \)) and \( \eta x \) chpos(\( r10, x, r7 \)) refOs r2 and r10 come up in the chain of grounding. Both contain no groundable designations, because the chpos descriptions are cyclic, and they contain no other designations. Consequently, both designations cannot be grounded and are not chosen.

The examination of the second refO (r9) yields that it has a directly grounded description, \( \eta x \) p_curved(\( x \)), which is chosen immediately, because it violates
none of the constraints. $\eta x \, \text{chpos}(r_{11}, r_1, x)$ is cyclic and, consequently, filtered out. The description $\eta x \, \text{transitionpoint}(r_8, x)$ leads to the investigation of $r_8$. Apart from the cyclic $\eta x \, \text{transitionpoint}(x, r_9)$, refO $r_8$ contains only descriptions with low activation values. The rationale is that the location is perceived mainly as a transition point and not as starting or endpoint, because at this point the orientation of the plane starts changing. Since the plane does not stop at this position, the latter two descriptions have a very low activation value. Thus, only if $at$ has a low value these descriptions can be used in the verbalisation. Let us assume that the activation is below the activation threshold ($\text{inC}$ sets these activations to 0.1), and no description of $r_8$ can be used for grounding. Then, the grounding of $\eta x \, \text{transitionpoint}(r_8, x)$ of $r_9$ fails.

Allow me short digression. Transition points seem to play a special role in describing motion events and are mostly verbalised if they can be specified by a landmark, which is not present in the example at hand. Note that this is a working hypothesis on the basis of the theoretical analysis I describe here. However, initial results from the corresponding verbalisation study corroborate this view. If the transition point is, for example, close to a tower, it can be referred to easily:

(4) *Das Flugzeug biegt am Tower ab.*
'The plane turns off at the tower.'

Furthermore, if the transition point were selected the resulting verbalisation would contain *abbiegen* 'turn off', as in

(5) *Ein Flugzeug biegt ab.*
'A plane turns off.'

Since the verb *abbiegen* 'turn off' incorporates the transition point in the underlying conceptual (semantic) representation, it is not necessary to verbalise it separately, eg by an additional phrase. Vice versa, the transition point must be present if *abbiegen* is to be generated, cf. Eschenbach, Habel, & Kulik (1999) and Eschenbach, Habel & Kulik (2000) for a formal analysis.

Finally, the description $\eta x \, \text{to}(x, r_5)$ is checked. $r_5$ is the left region of the reference system created by the straight path $r_3$. Therefore, it contains the functional expression $\text{Left} \_ \text{region}(r_4)$. $r_4$ refers to $r_3$, which contains the directly grounded description $\eta x \, \text{straight}(x)$. Thus, $\eta x \, \text{to}(x, r_5)$ is grounded and, hence, chosen. However, since $r_3$ is also part of a refO in the current preverbal message ($r_9$), the homogeneous-part-of constraint causes that none of $r_4$'s descriptions is actually chosen once it is actually generated. This is a case where a groundable designation is not chosen. In other words, $\eta x \, \text{straight}(x)$ serves to ground a designation ($\eta x \, \text{to}(x, r_5)$) but is not verbalised, because it is ruled out by another constraint.

With this the verbalisation of the situation represented by $r_{11}$ is finished. The preverbal message created this way will result in an utterance like the one given in (4), repeated here as (6).
'A plane moves to the left.'

b. 

\[
\text{situation} \rightarrow v_1 \quad \eta \ chaos(v_2, v_3) \\
\text{plane} \rightarrow v_2 \quad \eta \ plane(x) \\
\text{path} \rightarrow v_3 \quad \eta \ p\_curved(x) \\
\text{region} \rightarrow v_4 \quad \text{Left\_region}(v_5) \\
\text{ref\_sys\_spat} \rightarrow v_5 \\
\text{verb\_of}(r_11) \\
\] 

In order to obtain the other verbalisations mentioned in this section the following incremental preverbal messages must be generated. (7) can be generated by using an \( \alpha_T > 0.5 \), (8) by an \( \alpha_T = 0.4 \), cf also the system output in section B.3.

(7) a. *Ein Flugzeug fährt eine Kurve.*
'A plane follows a curve.'

b. 

\[
\text{situation} \rightarrow v_1 \quad \eta \ chaos(v_2, v_3) \\
\text{plane} \rightarrow v_2 \quad \eta \ plane(x) \\
\text{path} \rightarrow v_3 \quad \eta \ p\_curved(x) \\
\text{verb\_of}(r_1) \\
\] 

(8) a. *Ein Flugzeug biegt ab.*
'A plane turns oﬀ.'

b. 

\[
\text{situation} \rightarrow v_1 \quad \eta \ chaos(v_2, v_3) \\
\text{plane} \rightarrow v_2 \quad \eta \ plane(x) \\
\text{path} \rightarrow v_3 \quad \eta \ p\_curved(x) \\
\text{verb\_of}(r_9) \\
\] 

\[ \eta \ transitionpoint(v_4, x) \]
Let me conclude this example with two remarks. Firstly, in connection with
the issue of perspectivisation I proposed that the sequence in which the increments
are generated has effects on the syntactic structure. If, for instance, to(r7, Left-region(r4))
is generated directly after the situation refO, a topicalised utterance like

\[(9) \text{Nach links bewegt sich etwas.} \]

‘To the left something moves.’

will be the result. Currently, pvm-generation visits the refOs in the order in which
they occur in the designation that is just verbalised. This, obviously, cannot be the
general solution, because in this way verbalisation refOs and, consequently, the components of utterances will always be generated in the same sequence. Thus,
either a special computational mechanism must be introduced that establishes an
appropriate ordering, eg by introducing additional constraints, or the order in which the refOs are verbalised is simply random. The latter solution probably will
not be able to account for the empirical data. Nevertheless, even the solution involving additional computational mechanisms should not perform an explicit linearisation task for the reasons I already gave. Instead, the computations should depend on issues like activation, focus, or attention.

Secondly, which utterances are produced depends on which situation refOs are
chosen by selection. For example, a lower value for \( \tau \) (a shorter latency) has the
effect that pvm-generation takes out the head of traverse buffer earlier so that not
the whole situation is verbalised. The alternative selection strategy furthermore has
the effect that not the most complex situation is taken (r11) but the two simpler ones (r2 and r10). This results in the generation of two preverbal messages. A corresponding verbalisation is:

\[(10) \text{Ein Flugzeug fährt geradeaus. Es fährt nach links.} \]

‘A plane is moving straight forward. It is moving to the left.’

The preverbal messages underlying such an utterance is:

\[(11) \text{situation \quad v1 \quad } \eta x \text{ chpos}(x, v_2, v_3) \]

\[
\text{plane} \quad v_2 \quad \eta x \text{ plane}(x)
\]

\[
\text{path} \quad v_3 \quad \eta x \text{ straight}(x)
\]

* Although using unconnected utterances may sound unusual, it is in fact predominant in the verbalisations given in appendix C.
After \( v_3 \) the verbalisation of the first situation is completed, and \( v_4 \) starts a new preverbal message. The second verbalisation of the plane (\( v_5 \)) contains no designation, because \( \eta x \text{plane}(x) \) was already used in the first verbalisation (\( v_2 \)).

What happens next. The designations of a refO are closely related to lexical items. For example, the description \( \eta x \text{to}(x, r_5) \) corresponds to lexical items of prepositions specifying a goal. Yet, a direct correspondence between designation and lexical item is not always given: the lexical item for *abbiegen* ‘turn off’ contains not only a transition point but also a component for motion, corresponding to a propositional formula like \( \text{chpos}(r_{11}, r_1, r_9) \). Each increment of a preverbal message that is received by the formulator triggers a lexical access, the selection of a lemma to be precise, which is the first step in the generation of the syntactic structure. Which lexical item is accessed depends mainly on which designations are chosen. The details, however, are beyond the scope of this investigation.

Evaluation. The three resource parameters \( \text{doat}, \text{lotb}, \text{and lt} \) will be discussed extensively in the next section, where they exhibit stronger influences on the generated verbalisations. I will, therefore, mainly discuss how the parameter \( \text{at} \) and the used selection strategy cause different behaviour of \( \text{inc} \), ie how different incremental preverbal messages are generated. I will only use the activation values of descriptions and no activations of refOs. Figure[14,4] shows the incremental preverbal messages generated by \( \text{inc}^+ \) for \( \text{doat} = 0.5, \text{lotb} = 6, \text{lt} = 125, \text{at} = 0.6, \) and the alternative selection strategy, which retains the level of granularity of the selected situations as far as possible, cf figure[11,5].

* The first verbalisation adds the description to the used\_desigs list, cf section[13,2]. See section[13,3] on how other possibilities can be generated.
† \( \text{inc} \) is still under development. The current version, with which these simulations were carried out, is 0.2.2.
This is the PVM file of inC version 0.2.2 (build 077)
DoAT=0.5; LoTB=6; LT=125; AT=0.6
Domain: motionEvents
Selection Strategy: retainGranularity
Real-Time: true; Time Frame: 50

New PVM starting with refO 8

situation ----------- r13 ---- eta x: chpos(x 14 15) <<0.5>>
verb_of(8)
at_time(0 200)
status(regular)
pe(2)

object ----------- r14 ---- eta x: plane(x) <<0.9>>
verb_of(7)
plane
status(regular)
pe(1)

path ------------- r15 ---- eta x: straight(x) <<0.6>>
verb_of(9)
status(regular)
pe(3)

New PVM starting with refO 16

situation ----------- r21 ---- eta x: chpos(x 22 23) <<0.5>>
verb_of(16)
part_of([19])
met_by(8)
at_time(201 320)
status(regular)
pe(7)

object ----------- r22 ----
verb_of(7)
plane
status(regular)
pe(1)

path ------------- r23 ---- eta x: curved(x) <<0.6>>
verb_of(17)
part_of([20])
status(regular)
pe(8)

Figure 14.4: An incremental preverbal message generated by inC for DoAT = 0.5, LoTB = 6, LT = 125, AT = 0.6
Since the numbers of refOs are assigned by the system, the numbering in the following differs from the one used above, and can, furthermore, differ in each simulation. In this notation <<...>> stands for the activation value of a designation. The attribute \texttt{at\_time} represents the time interval of the situation, eg \texttt{at\_time(0 200)} says that \( r8 \) happened in time frames 0 to 200. Since each time frame spans 50 ms, this means that \( r8 \) lasted 10 s. Two situation refOs, \( r8 \) and \( r16 \), are selected in this simulation. Since this means that they must be verbalised, their designations are selected although the activation is below \( \alpha_t \), cf chapter [13]. For the other refOs, there are designations above \( \alpha_t \). The incremental preverbal message may result in an utterance like

\begin{quote}
\textit{Ein Flugzeug fährt geradeaus. Es fährt eine Kurve.}
\end{quote}

'A plane moves straight on. It moves on a curve.'

For \( \alpha_t = 0.5 \) the incremental preverbal message is continued as shown in figure [14.5]. The full output of this and the following simulations is given in section [15.2]. In this simulation the \texttt{eta x: to(x, 24)} is chosen by \texttt{rvm}-generation as well, which means that \( r24 \) and \( r25 \) must also be generated. The output is similar to the one in [6].

Another verbalisation arises if the standard selection strategy is used so that the first incremental preverbal message – describing the straight movement – is identical to the one above. For the second preverbal message, however, the situation refO representing the whole movement is selected (\( r19 \) in the output given in figure [14.6]). The output given in figure [14.6] uses an \( \alpha_t = 0.4 \), but the same output is generated for \( 0.1 < \alpha_t < 0.5 \). The difference here to simulations using the same selection strategy with a higher value of \( \alpha_t \) is that \texttt{eta x: transitionpoint(24 x)} is generated as well, which will have the proposed impact on the selected lexemes.
14.2 A WORKED EXAMPLE

--- New PVM starting with refO 19 ---

situations ----------- r21 ---- eta x: chpos(x 22 23) <<0.5>>
verb_of(19)
parts([8 16])
at_time(0 320)
status(regular)

object --------------- r22 ----
verb_of(7)
plane
status(regular)
pe(1)

paths --------------- r23 ---- eta x: transitionpoint(24 x) <<0.4>>
verb_of(20)
eta x: p_curved(x) <<0.6>>
concat([9 17])
status(regular)

locations ---------- r24 ----
verb_of(18)
status(regular)
pe(9)

Figure 14.6: The second incremental preverbal message for \( \alpha_T \) = 0.4 and the standard selection strategy

later on: with this additional designation a lexeme like *abbiegen* ‘turn off’ is chosen.

(13) *Ein Flugzeug fährt geradeaus. Es biegt ab.*
‘A plane moves straight on. It turns off.’

If \( \alpha_T \) is set to 0.1 even more designations and refOs are generated. The (rather strange) output is given in section 5.2. It must be doubted that these incremental preverbal messages do indeed reflect the human verbalisations. A possible interpretation would be that if humans do not know what to say, they simply say everything they can.

Summing up, for the discussed example, two factors mainly influence the generated output. The selection strategy decides whether the two simple situations are described, or the first of the simple ones and the complex situation. If additionally \( LT \geq 6050 \) ms only the complex situation is generated. The value of \( \alpha_T \) determines which designations are used in the incremental preverbal messages; \( \alpha_T > 0.1 \) ensures that no spurious designations are chosen.
I already indicated that the first phase of developing \textsc{inc} was not done in the domain of motion events but in the domain of the verbalisation of dynamic sketch maps. For this reason I want to describe a simulation within this setting in this section. Additionally, descriptions of sketch maps have the advantage that the event structure is more complex, which means that the effects of different values for the three parameters \textsc{doat}, \textsc{lotb}, and \textsc{lt} can be demonstrated more easily. The simulations reported in the following were carried out with a precursor of the current implementation, which was done in Prolog. One insight gained by these simulations was that Prolog needs a lot of resources for the algorithms used in \textsc{inc} – run-time as well as memory. This was the main reason that \textsc{inc} now is implemented in Mozart/Oz, which supports the process-based architecture much more directly. As a consequence, the current version of \textsc{inc} is much more efficient. The older implementation of \textsc{inc} also contains no parameter \textsc{at}. Yet, since this parameter is mainly used in the selection of designations in \textsc{pvm}-generation – which is not done in the following –, this has no effects on the data discussed here.

For this domain drawings of sketch maps were recorded in a first phase of the empirical studies with a drawing tablet. These sketch maps were then presented to participants of verbalisation studies, who were instructed to describe what they saw. The sketch maps were shown to them as they developed on the screen. In technical terms: each 50 ms a pixel was added where the drawing pen was registered while the drawing was recorded. (It was also recorded whether the pen was up or down. Accordingly, a white or a black pixel was added.) The important point is that the participants did not describe a sketch map as such but its \textit{drawing}. The reason, again, is that the project ConcEv, where these studies were carried out, is about investigating the conceptualisation of events.

An example of a very simple sketch map is shown in figure 14.7. It is a crossing consisting of eight lines. The numbers in figure 14.8 indicate two different sequences in which the eight lines of this crossing were drawn. A closer look reveals that the two crossings also differ in the shape of the lines, eg the lines annotated by 1 and 5 in...
the right crossing do not meet at their endpoints, while in the left drawing they do. These differences are due to the fact that the sketches were drawn free-hand. Since such deviations from ideal straight lines are handled by the \texttt{ppu} for this domain, they will concern us no further. On the conceptual level they can be regarded as ideal lines. The crucial point here is that the lines are read in by \texttt{inc} in different sequences. Thus, although the object structure is identical in both cases, the event structure is not. In the following I will mainly consider the left crossing.

The hierarchical event structure of the left crossing is given in figure \ref{fig:crossing_event}. In this figure \texttt{dr-hl} stands for the event of drawing a horizontal line, \texttt{dr-vl} for drawing a vertical line. \texttt{dr-par} represents the drawing of two parallel lines, eg the ones annotated by 1 and 8 in figure \ref{fig:crossing_lines}. \texttt{dr-cor} that of a corner, ie two lines meeting at their endpoints, eg 1 and 2. \texttt{dr-crossing}, finally, denotes the event of drawing the whole crossing. In figure \ref{fig:crossing_event} the line from \texttt{dr-par4} to \texttt{dr-hl1} is left out for better readability. The temporal order of the events corresponds to their positions from left to right. The structure of the object representation is isomorphic to the event representation, except for the temporal ordering.

The sketches were presented to twelve participants. Their verbal descriptions were recorded and transcribed. Ten of the participants used the following pattern for the left crossing:

![Figure 14.8: Different sequences of drawing the lines of the crossing](image)

![Figure 14.9: The event structure of the crossing](image)
Figure 14.10: Log-output for the verbalisation of the crossing

- first segment (DR-HL1)
- intermediary complexes (DR-COR1, DR-PAR1, DR-COR2)
- crossing expected (DR-CROSSING)
- intermediary complexes (DR-COR3, DR-PAR3, DR-COR4)
- crossing complete (DR-CROSSING)

One of the other two kept silent until the sketch was complete and then said:

(14) *eine Kreuzung oder 'n Kreuz*  
'a crossing or a cross'

The twelfth uttered no expectations but only described the emerging lines, before naming the result. The other 10 differed only slightly in how much of the crossing was visible when they produced the expectation, but all named it before it was fully visible. The verbalisations of the other crossing stick to a similar standard pattern:

- the first two segments (DR-HL1, DR-HL2)
- intermediary complexes (DR-PAR1, DR-PAR2, DR-COR1)
- crossing expected (DR-CROSSING)
- intermediary complexes (DR-PAR3/DR-COR2, DR-PAR4/DR-COR3)
- crossing complete (DR-CROSSING)

The output of a simulation for the first crossing with DOA = 0.5, LOTB = 6, and LT = 125 ms is given in figure 14.10. The following notation is used. Lines starting with TB contain the state of the traverse buffer at that point in time. They were generated as soon as the content of the traverse buffer changed. Lines starting with Gen

* Here, DR-PAR and DR-COR are generated at the same time. Since there is no temporal criterion for selecting one of the two, the choice is random.
indicate that this refO is verbalised, ie sent to the formulator by pvm-generation. Entries like hline1 stand for the corresponding node in the event structure, here: DR-H11. crossingE is an expected crossing, crossingR a completely perceived one. So, here preverbal messages are not generated completely but only their ‘core’. For example, Gen: hline1 can be understood as a preverbal message that leads to an utterance like

(15) *Ein horizontaler Strich wurde gezeichnet.
‘A horizontal line has been drawn.’

Evaluation. The output of the simulations is close to the observed human verbalisations. In other words, the sequence of preverbal messages (Gen lines) corresponds structurally to the sequences of utterances of the human verbalisations.

The values for LT and the run-time of the program in the following are machine dependent and possess only limited cognitive adequacy, because this implementation of INC only reads prefabricated input files as well. More importantly, however, this implementation did not have the ability to simulate real-time behaviour. Thus, the simulations were faster than real-time, albeit directly proportional to it. The values for DOAT and LOTB certainly come closer to being psychologically real.

DOAT was set to 0.5 and 0.9, LOTB to values between 2 and 6, and LT was varied between 50 ms and 2000 ms. The run-time of the program is split into two phases: 1400 ms for the initialisation of program and the concept storage, and the run-time of the conceptualisation. The second value, which is the important one here, is 2700 ms to 3000 ms for the first crossing and 2500 ms to 2800 ms for the second one. The differences in run-time are due to the fact that several pre-processing files per sketch were used, all of which yielded equivalent results but took a different run-time. The values of the parameters for both crossings resulting in the simulation of the human verbalisations sketched above were determined as INC’s default values: DOAT = 0.5, LOTB = 6, LT = 125 ms. Varying these values creates the following effects.

The maximum DOA for the crossings before they are complete is 0.81. Therefore, the maximum value of DOAT that has the effect that an expected crossing is introduced into the ccr is 0.8. Thus, using DOAT = 0.9 instead of 0.5 means that no expectation is inserted into the ccr.

A value for LOTB > 5 is never needed. LOTB = 5 is required for LT ≥ 800 ms, if no elements of the traverse buffer shall get lost. LOTB = 3 suffices for most settings; only with LT ≤ 125 ms a value of 2 suffices. Other settings (LOTB < 3 with 125 ms < LT < 800 ms or LOTB < 5 for LT ≥ 800 ms) have the effect that elements are lost. A value of LOTB = 1 means that elements are lost, and in almost all cases only the complete crossing is verbalised.†

* In fact, each value of LT > 1 suffices for the default case, but this way it is ascertained that no element of the traverse buffer gets lost.
† In this case the output sequence shows considerable gaps, which corresponds to verbalisations in which the participant did not keep up with the ongoing events. In the recorded verbalisations, however, there were no such cases.
As in the case discussed in section 14.2, decreasing the value for $LT$ means that more preverbal messages are produced, increasing it leads to the production of less. From an $LT \geq 800$ ms on no simple elements and no intermediary complex refOEs (parallel, corner) are verbalised but only the expected and the complete crossing (crossingE and crossingR). The maximum value that still leads to the production of a preverbal message for the expected crossing is ca 500 ms for the first crossing and ca 1000 ms for the second crossing. So, using these values produces the verbalisation of participant 11 given in (14). With $LT < 125$ ms more simple events and some intermediary complex ones, which are otherwise replaced by selection before they are generated, are verbalised. With $LT = 50$ ms each selected node is verbalised. This is the pattern observed with participant 12.

Thus, 1NC produced all types of verbalisations present in the verbalisation corpus. The different verbalisations are produced by varying the parameters. The main class of verbalisations is produced by setting the parameters to their default values. (Vice versa, the default values were determined by evaluating the corpus.) But 1NC can also generate the more unusual cases of participants 11 and 12.

---

* Both values should be more or less identical. The gap is an artefact that arises, because the simulations were carried out with use pre-processed files as input.
MONITORING is the second component besides message generation in the conceptualiser of Levelt’s (1989) model of the language production system, cf figure 5.2. It compares actual utterances of the speaker with the planned ones. That is, it compares the output of the language comprehension system for the speaker’s own utterances to the utterance plans. InC mainly is a model of the message generation component. However, there also exists a rudimentary implementation of a monitor extending this component, cf figure 5.1.

The difficulty in developing a detailed monitor for InC is that it depends on artificially generated errors, because there is no implementation of the formulator and no language comprehension system. And, figuratively speaking, one cannot be certain that detecting such errors is not just pulling the rabbit out of the hat one has put into the hat just before. Therefore, the goal of this prototypical component mainly is to evaluate the theoretical implications on InC. In the standard mode of operation the monitor is not running.

In conceptualisation there are two main sources for errors: performance errors, usually detected by self-monitoring, and what I call conceptual changes (Guhe & Schilder 2002a). Performance errors are errors that are caused by a malfunction of the system. For example, the wrong lexeme can be accessed after lemma selection. Conceptual changes are changes in the conceptual representation that affect content that has already been uttered, more precisely: that has left the conceptualiser and been sent to the formulator. While for performance errors it can be argued that the goal in building NLG systems should be that the systems do not make such errors† it is impossible to avoid conceptual changes in an incrementally working system. Two methods of error detection must be distinguished:

* Calling InC with the monitor in operation requires that the loop from preverbal messages to parsed speech is somehow closed. Since there is no implementation of formulator and language comprehension system, the monitor cannot be activated in the simulator window shown in figure 6.1. Instead, the InC package is called directly from another Mozart /Oz program that builds this loop, ie it reads the preverbal messages generated by InC, inserts errors, and sends this output back to the InC monitor as ‘parsed speech’.

† This argument is usually accepted by non-cognitively motivated NLG research.
1. comparing planned and actually produced utterances,
2. comparing planned (and/or produced) utterances and the current state of affairs.

While the first method is exclusively performed by the monitor, the latter can be used at different points of conceptualisation by different processes:

1. PVM-generation compares the content of the current preverbal message to the content of the CCR, cf chapter [13] If a refO in the CCR changes that is used in the current preverbal message a self-correction is generated.
2. PVM-generation detects 'dead ends', ie situations where the speaker 'talked himself into a corner' (De Smedt, 1990a:29). Yet, PVM-generation can only do this for semantic errors. If the dead end is due to the grammatical structure a cooperation of monitor and language comprehension system is required to detect it.
3. Construction can capture the most general case. It can compare the content of the traverse (or the rest of the CCR for that matter) with its focussed element. Remember that the focussed element of construction is the element of the CCR that was either perceived most recently or that was involved in the last inferences or deductions. If it turns out that a belief that was held to be true up to that point or that was already
uttered has to be revised, then a correction can be constructed. In this case the contradiction must be explicitly represented in the ccr in order to be selected and verbalised.

4. The monitor can compare what was said (what is in the traverse) with the current state of affairs (stored in the ccr).

Only two of these cases are implemented. The monitor compares planned with actually produced refOs (case 4), and pvm-generation compares the refOs in the current preverbal message with the current state of these refOs in the ccr (case 1). The functioning of the monitor given in the following depends on the notion of the monitored traverse.

Term 15.1 Monitored traverse. The monitored traverse is the part of the ccr that contains the information that has been received by the monitor from the language comprehension system. This information is correlated with the information in the ccr and the traverse. In the ideal case this is a one-to-one correspondence of refOs.

The correlation between refOs in the monitored traverse and other refOs of the ccr, in particular those in the traverse, is established via computing similarities between monitored refOs and refOs of the ccr. Thus, if there are no errors each refO in the monitored traverse has a corresponding refO in the traverse.

Figure 15.2 shows the overall structure of the monitor process. The monitor process loops until it is certain that no more parsed speech will follow that corresponds to preverbal messages generated by the message generation component. Technically, the monitor and pvm-generation follow a synchronisation protocol that ascertains that the monitor will send not more instructions to pvm-generation to generate a correction and that equally makes sure that pvm-generation does not generate any output that the monitor will have to check against the content of the traverse. Yet, the details are of no interest here.

In the loop the monitor waits for an increment of parsed speech. When one is available it is appended to the monitored traverse. It is checked whether the increment has a corresponding increment in the traverse or whether, more generally, a new mismatch between monitored traverse and traverse can be found. If an error is detected, the message generation component is suspended and a signal is sent directly to formulator and articulator. This is done, because humans are able to stop utterance production almost instantaneously upon detection of an error \citep{HartsuikerKolk2001}. Then, the correction is computed, inserted into the

* The knowledge that is corrected by such a verbalisation needs not be uttered already – paradoxically as it may sound in this context. An example is:

(i) *The ball is red. I thought it was orange.*

However, this no real self-correction in the sense used in the rest of this work.

† This really is the thesis that language comprehension is language production ‘backwards’ in disguise. And in an ideal communication the ideas that are produced in the hearer’s head are identical to the ones in the speaker’s.
MONITORING AND SELF-CORRECTIONS

```plaintext
var monitored_traverse,
    traverse, #traverse in the ccr
parsed_speech_stream, #input stream containing the parsed speech
pvm_gen_stream  #increment stream for coordination with pvm-generation

while coordination for termination with pvm-generation fails do
    wait parsed_speech_stream ≠ ()
    input_inc ← Fetch(parsed_speech_stream)
    Append(input_inc, monitored_traverse)
    if input_inc ∉ traverse or
        Mismatch(monitored_traverse, traverse) then
            suspend message generation component
            signal [formulator, articulator]
            correction ← ComputeCorrection(input_inc)
            Insert(correction, ccr)
            Append(correction, pvm_gen_stream)
            resume message generation component
    endif
endwhile
```

Figure 15.2: The monitor process

corr, and sent to pvm-generation. Finally, the message generation component is
resumed. Note that I did not describe the corresponding part of pvm-generation
in chapter 13. pvm-generation checks at the beginning of each iteration of its loop
whether the monitor sent a correction. If this is the case it generates the required
increments.

Yet, the current implementation is simpler than this. The monitor only tries to
find a refO in the traverse that was not yet received as parsed speech and that is
identical to the input increment (input_inc). If this fails, ComputeCorrection
searches for the refO that is the one most similar to input_inc. This refO is as-
sumed to be the refO that should have been verbalised. The correction informa-
tion sent to pvm-generation consists of

- which refO to correct,
- which information is to be deleted from refO, and
- which information is to be added to refO.

Especially one problem is left open by this solution: the parsed speech does not
have to be subdivided into refOs in the same way as the preverbal messages that
were generated. (After all, language comprehension is not simply language produc-
tion backwards.) Hence, finding correspondences between monitored traverse and
traverse is quite a complex task.

Finally, with regard to incremental generation the interleaving of production
(message generation) and monitoring (comparing, controlling, and generating self-corrections) is a very important aspect. The central question for a cognitively adequate model is how far apart conceptualisation and reception of the corresponding parsed speech are. This can be asked in relative as well as in absolute terms. The former question is what time intervals (how many milliseconds) there are between the generation of increments of preverbal messages and receiving the corresponding increments by the monitor. The latter question is how many increments message generation – PVM-generation in particular – is ahead in comparison to the corresponding parsed speech. Answering these questions hinges on the following points (both of which can be understood relative as well as absolute):

1. How long does it take for an increment of an incremental preverbal message to be encoded into language and decoded again into parsed speech?
2. How long does it take for an increment to be generated? In other words, at which rate does the conceptualiser generate increments of preverbal messages?

It probably is a good guess to assume that both proceed at more or less the same speed. But these are difficult questions, in particular because what increments are is hardly empirically tractable so far up in the language production system. However, [Hartsiaker & Kolk (2001)] present empirical evidence supported by a computational model that give first clues. They will have to be taken into account in the further development of rsc, but I will not discuss this issue here. All in all, though, it can be said that self-corrections are generated rapidly. Often utterance production (including articulation) is interrupted while the flawed segment is uttered [Levelt (1983)]. Thus, the feedback by the parsed speech is quickly available, and there is only very little delay between generating an increment of a preverbal message and – in case it is immediately generated by the formulator, ie no reordering takes place – perceiving the corresponding verbalisation. At least the time needed is short enough so that generation of an increment, parsing it again, and interrupting ones own speech can all occur during the production of the increment.
RESULTS
LIMITATIONS, MODIFICATIONS, AND ENHANCEMENTS OF inC

Models like inC can never be complete. Some of inC’s limitations and possible enhancements were already discussed in the text. Instead of giving an overview of future research I want to discuss the question of what this investigation has archived and what is still missing by looking at the current state of inC. In section 16.1 I will, therefore, first present the major limitations of the version of inC that I described in part C. In section 16.2 I consider the most interesting modification of the approach I have taken, viz using procedures instead of processes as main means of structuring. Finally, in section 16.3 I propose some necessary and/or insightful enhancements of the model, in particular some additional parameters to influence its behaviour.

16.1 Limitations

Let us first look at some limitations of inC in its current state. While the unidirectionality of the information flow is one of inC’s strengths, because it reduces the complexity of computations, there is one place in the model where it severely limits its cognitive adequacy: inC does not give feedback to the PPU. This makes it impossible to account for phenomena in which conceptualisation influences perception. Conceptualisation influences the visual sub-system, for example, so as to control the eye-focus. That this is required by a cognitively adequate model is shown in experiments like the ones carried out by Griffin & Bock (2003) and Bock, Irwin, Davidson, & Levelt (2003).

The high efficiency of inC, which is mainly due to the unidirectionality of the information flow, comes at a price. The fixed arrangement of processes and the non-use of feedback in the cascade makes inC rather inflexible. Nevertheless, even if more flexibility of the architecture were required for other tasks, eg more feedback, the arrangement of processes in inC’s cascade that execute the four tasks would have to be performed in the same sequence: only what has been inserted into the conceptual representation can be selected for verbalisation, only what has been selected can be linearised, and only then can a preverbal message be generated.
Additionally, it is not possible to generate multiple utterance candidates from which the best one is chosen for verbalisation. Equally, a revision-based mode of operation is not possible, i.e., it is not possible to generate an utterance, receive it from the language comprehension system, and evaluate whether the communicative intention to be conveyed is expressed in an adequate quality. With regard to the cognitive adequacy of \( \text{INC} \) these would be interesting alternative modes of operation, because they are obviously used by humans, e.g., when writing scientific texts. In order to do this, however, models of the other components (formulator, articulator, language comprehension system) are required. Then, it could be evaluated which of \( \text{INC} \)'s current mechanisms can be (re-)used, whether they would need to be extended and/or combined differently. This would be a step towards a more complete model of the conceptualiser.

\( \text{INC} \) is a model with bounded rationality. One of the consequences is that it carries out its operations on local contexts in a limited amount of time. The former means that not all available knowledge is considered, the latter that not all possible inferences are drawn, because Extended Wundt's Principle (massive incrementality) is an early commitment strategy. Since the scenes I considered here were rather simple, there was no need to equip \( \text{INC} \) with capabilities for resolving inconsistencies that arise as a consequence of this. Nevertheless, such mechanisms must be incorporated. As this could have serious repercussions on \( \text{INC} \), it should be done as a proof that the overall model is valid.

Finally, \( \text{INC} \) does not account for the domain-dependency of cognitive performance. Although this did not cause problems so far, it should be regarded a limitation of the model. Furthermore, \( \text{INC} \) only verbalises events (situations). Although the description of objects or other entities should work along the same lines this has not been corroborated.

### 16.2 Modifications

\( \text{INC} \) is a process model. Based on the points raised in the previous section one should consider the possibility that a fixed cascade of processes in the mind may not be the best way to model the conceptualiser. A modification of \( \text{INC} \), therefore, consists in replacing the processes by procedures that are called from a central instance. That is, instead of having processes that run continuously and suspend themselves when they have nothing to process, one could use procedures that are called from a permanent instance and that free the resources they use when they terminate.

However, two of the central ideas of incremental processing I proposed, namely cascading and using one representation for the conceptual representation, should be preserved in this approach. Thus, firstly, the information flow between the components that are currently modelled as processes must be retained. For example, it must be warranted that when construction inserted a new element into the CCR selection is called to evaluate whether this new element is to be chosen for verbalisation. Secondly, these procedures must not contain internal storages, because the
stored knowledge would get lost when the procedure terminates. This is particularly true if multiple instances of procedures are allowed, see below. (Alternatively, this knowledge could be passed on to the next call of the procedure, but this is too awkward a solution.)

Since the parallelism (concurrency) of inc should be preserved, the solution of using procedures has the disadvantage that it must be checked whether there exists more than one instance of a procedure at a given point in time. This is, for example, necessary to avoid racing (‘last-writer-wins’) situations, cf page 99 – except if this kind of indeterminacy adds to the cognitive adequacy of the model. Since procedures do not possess the coordination mechanisms of processes, this must either be controlled by an additional component, eg a central dispatcher, or corresponding checks must be made before a procedure is called.

There are cases, though, where multiple instances of a process (procedure) are advantageous, cf page 104 on processing multiple increments in parallel. Yet, it must be distinguished whether the instances work on the same increment or on different ones. The latter is used, for example, by De Smedt (1990) and can indeed enhance the cognitive adequacy. In this case a new instance is spawned when a new increment becomes available. This method provides explanations for phenomena in which simpler increments are generated first although they were read in later. If all instances work on the same increment different versions of a procedure, eg multiple selection strategies, can be tried in parallel. Yet, as also pointed out on page 104, this includes the necessity for a frame process (procedure) that encapsulates the instances, because one of the results must be determined as overall result.

Parallelism of this kind is especially useful in models with more than one input stream. An example for inc is if the input is not only visual but also auditory, which enters the system via another input stream. Then, in addition to the parallelism described so far, it would be advantageous to have more than one process for construction. Similarly, if the conceptualiser provides information in multiple output streams a process corresponding to pvm-generation should be used for each stream, eg for the generation of gestures to take the most obvious example.

16.3 Enhancements

The first real enhancement of inc must be to develop and implement the linearisation process. Despite the fact that it is the most dispensable of the four main tasks, it is important and, furthermore, there is a substantial amount of literature on this topic, cf section 5.2.

Similarly, since inc is a model of the message generation part of conceptualisation only, the monitor should be added, which I already did up to a certain degree in chapter 15. Monitoring is especially relevant, because self-corrections are a valuable source for investigating conceptualisation issues for which – as for linearisation – there is already a substantial amount of research.

The next significant enhancement is that inc must be able to detect decision
cues in the represented knowledge. For example, I described two different selection strategies in chapter[11] Applying them both at the same time only shifts the selection problem: the result of which algorithm is taken? Instead, it is desirable to decide upon one strategy on the basis on cues that are obtained from the environment or the current state of the model. Alternatively, it may be possible to integrate both selection strategies, but then cues must be used as well in order to decide whether to retain the level of granularity or to use the most complex concept not selected for verbalisation.

_Learning_ is often seen as a byproduct of reasoning [Markman & Gentner 2001]. This is even one of the main tenets of unified cognitive architectures. Thus, it can be sensible to treat the extraction and detection of decision cues as a learning issue. For example, the early detection of complex concepts can be learned on the basis of how successful the generation of expectations was. (What cues lead to a correct guess?) Apart from this, concept learning may be an enhancement that gives new insights into the functioning of the conceptualiser.

A related issue is that new heuristics should be developed and/or existing ones should be extended. One possibility is to apply the 'simple heuristics that make us smart' [Gigerenzer, Todd, & the ABC Research Group 1999; Todd & Gigerenzer 2000] to the tasks of conceptualisation. Since these are general decision-making heuristics, the transfer to conceptualisation must be performed with great care. Furthermore, the term heuristic as it is used by Gigerenzer et al is more like what I called selection strategy than the heuristics I used for establishing a local context. Similarly, anC only has a very rudimentary activation model, which is usually an important issue in cognitive models. Apart from a precise specification of when the activation constraint in PVM-generation can be violated the static activation model must be replaced by one that adequately integrates decay and reactivation. Additionally, I only hinted at the possibility of modelling priming effects by pre-activating elements of the available knowledge that become the local context for the next execution of the incremental algorithm, cf page[82] Since priming usually is a very revealing issue in cognitive models, it is a major source for new insights. Yet, for this to be possible, an activation model must be incorporated first. This is also true for enhancing anC’s cognitive adequacy by not always deleting the head of traverse buffer when the buffer is filled and a new element is appended. Instead, the element with the lowest activation should be deleted, cf the footnote on page[166].

Some of anC’s limitations can be overcome by extending it to a full agent model, for which the notion of an incremental agent could be used, cf pages[100–101] (According to the classification of agents in Luck & d’Inverno 2001 anC in its current form is a deliberative agent, which is rather low in the 'agent hierarchy'.) The main advantages would be that an agent is much more autonomous and that it provides the groundwork for two major enhancements. Firstly, multiple instances of anC can interact with each other, in particular, they can communicate on the level of preverbal messages. Since preverbal messages are represented as referential nets they can be read in and processed easily by a modified construction process. Secondly, anC can not only react to the environment but actually act in it. For example, it
could navigate through the environment on the basis of a conceptual representation similar to the representations used for the description of motion events.

Another big enhancement along the same lines is to integrate other modalities than language. In such a multimodal model the **C** is used not only for the generation of preverbal messages but for actions in other modalities as well, cf also [Reithinger 1991, 1992] and [Habel 2003]. Of particular interest is the generation of gestures as was already pointed out in the introduction. Previous work in this direction has, for example, been carried out by [van der Sluis & Krahmer 2001], [Krauss, Chen, & Gottesman 2000], and [de Ruiter 2000].

Finally, **C** could and should be extended by more parameters. The aim in developing a cognitive model should be to make claims about cognition that are as strong as possible. This is sometimes taken to mean that the model should have only few parameters with which it can be adjusted to fit empirical data. However, the parameters in **C** have a different function, namely to show that different factors influence its behaviour. Thus, these factors are pulled out into the open what otherwise might be kept hidden in the functioning of the model. For example, a parameter like **pvm idle time** described below can model the time span that a speaker can ‘bear it’ to keep silent. To model this explicitly is quite different from finding values for the available parameters so as to evoke the desired behaviour. For instance, if **C** shall ‘talk’ continuously in order to simulate a certain type of verbalisation, it would be equally successful to use a value for **pvm** high enough to ascertain that there always is something to be verbalised. Equally, **pov** can be increased so that output is slow (but not so slow as to produce silences longer than the maximal **pvm idle time**), or it is possible to lower **doat** so as to generate a lot of (mostly implausible) expectations. All of these methods obscure the real aim, viz to produce output without gaps longer than the maximal **pvm idle time**.

**Length of pre-process buffer.** The pre-processing unit works incrementally as well. Assuming a fixed time interval in which the **ppu** scans the observed environment, eg the standard 50 ms interval, this parameter determines how much of these intervals can at most be stored until they need to be given to **C**; ‘at most’, because when an event is finished the **ppu** should in every case pass on to **C** what it has collected up to that moment. So, this parameter influences at which rate the **ppu** sends new **perceived entities** to **C**. (Since there shall be as little feedback in the model as possible, this implies that preferably **C** cannot influence this parameter.)

**Length of perceived entity buffer.** An increment buffer between **ppu** and **C** has the advantage that both components can operate with greater independence of each other. This parameter determines, how many elements can be stored in the buffer. (It may, however, be a technical necessity only without cognitive counterpart.)

**Expectation retention threshold.** Construction can be equipped with a reluctance to change an existing expectation. For instance, if the previous **doa** was 0.6 and the actual is 0.63 but for a different best match, it may be useful, not to change the exist-
ing expectation but to retain the existing one. For the example a value of $\text{ERT} \geq 0.03$ would have this effect. The motivation for this parameter is that changing expectations is costly in terms of resources.

*Fill rate of traverse buffer.* A parameter like this can be integrated into selection in order to keep the traverse buffer filled to a desired degree. Consider that the traverse buffer contains two elements and that this parameter is set to 2. If then the two elements were to be replaced by a more complex one (according to the standard selection strategy) this parameter value would prevent the operation so that the traverse buffer retains the fill rate of 2.

*Event thread retention.* As already pointed out in chapter 12, one of linearisation’s main functions should be that concurrent event threads are not described in a permanently alternating fashion, but by describing one thread for a time, before describing another one. This parameter models the preference to stick to one event thread. It should be the first one modelled when the handling of concurrent motions events is fully integrated into inC.

*Output refO rate.* Producing language fluently means that the parts (syllables, words, phrases, sentences, utterances) are generated at a relatively constant rate. To induce the formulator to work at a constant speed the conceptualiser should produce output increments with a steady rate. Therefore, this parameter determines the target rate with which refOs are given to the formulator by $pvm$-generation. A second parameter can state the maximal deviation from this value.

*Maximal $pvm$ idle time.* This parameter models a point similar to the previous one. It specifies the maximal time interval that $pvm$-generation produces no output increment. When this time span has passed $pvm$-generation *must* generate output even if the $\text{IT}$ for the head of traverse buffer is not over. The requirement can even be made stronger in that output must be produced even if the traverse buffer is empty. In this case $pvm$-generation can produce a hesitation or a fixed phrase like *Nothing happens right now.* (This phrase would, of course, only work for the domain of event descriptions.) The parameter can also be located in the selection process.

*Idle time of processes.* If an increment stream or an increment buffer is empty when the reading process tries to retrieve an element, the process has three options:

1. continue processing available information so as to improve previous output,
2. suspend itself until the next element is put into the channel by the writing process,
3. suspend itself for a certain idle time.

If the last method is realised instead of the other ones, which are used now, the parameter determines the suspension time of the process. This method requires less supervision of increment streams and buffers, which speeds up processing.
To conclude this investigation I want to summarise the main points and put them into the general context again. I have explored to a certain depth the phenomenon of incremental conceptualisation for language production. My goal was to get from visual input to semantic output. I suggested an answer to some questions, different ones have been left unanswered, and plenty of new ones have arisen. I suspect that at some points you disagreed with my choice of generality or specificity in which I treated a topic, and very likely you missed a discussion of your favourite author, paper, or research programme. Yet, my goal was to make a first attempt to go all the way through the huge area called conceptualisation, not only to consider issues of decision making or categorisation, to name just two important ones relevant for conceptualisation. (Let alone the tricky issues of intentionality or consciousness.)

Separating conceptualisation from other cognitive faculties is difficult, because the components by which its tasks are carried out are modular only to a certain degree and, therefore, only partly independent of other cognitive faculties. Hence, I characterised the components of conceptualisation as quasi-modules, which possess the domain specificity of modules but lack their strict informational encapsulation.

The reduction to an online setting provides the means to boil down the endless complexity of conceptualisation to a degree where it is possible to account for the whole stretch with a computational model. The advantage of the online setting is that the temporal interleaving of processing perceptual input and generating verbal output allows to correlate input and output, despite the major difficulty in investigating conceptualisation: it is never directly observable but only via systems lying closer to the surface of cognition. In most cases this is language. Nevertheless, conceptualisation is a task in its own right, ie independent from language. Firstly, generating communicative intentions or sub-intentions, as I called the particular kind modelled by intC, is no language specific task; language is merely the means by which this particular type of intention is expressed most often. But there are others, like actions or gestures. Secondly, conceptualisation is a mediator between language and other cognitive faculties. When we want to talk about what happens
in the world, we perceive them in a modality different from language first. Conceptualisation bridges the stretch between non-linguistic representations and preverbal messages. (Although the perceived entities that are the input for inC are in propositional format, they are no linguistic representations.)

The online setting had two additional advantages. Firstly, it allowed to focus on the conceptualisation of events in contrast to the usually investigated conceptualisation of objects. Events, just like objects, are organised in hierarchical structures, in a subsumption (is-a) hierarchy and in a part-of hierarchy. Secondly, it was easier to investigate the difficult task of deciding what to say. Giving online descriptions of events requires that utterances are produced at a rather constant rate. Yet, the point of utterance initiation has strong influences on the generated verbalisation, because only the knowledge available at that point of time can be used (no future knowledge is available), and only the ongoing event or those recently finished are considered for verbalisation (the speaker has to keep up with what is happening).

The time pressure that comes with the online setting can be coped with elegantly by an incremental mode of processing. Thus, it is a way to cope with limited resources. It can deal with the time pressure, because incremental processing considers only the changes, not the complete representation. Since this also allows to identify a focussed element in the representation, viz the item that was processed last, the knowledge that is considered in the computations can be reduced to the knowledge connected to the focussed element. I called this knowledge the local context.

Based on this and on different kinds of incrementality that are proposed in the literature – some of which are only remotely related to incrementality as I laid it out here – I provided a definition of incrementality. The two most important terms in this definition are incremental model and incremental behaviour. An incremental model is a computational model consisting of a cascade of incremental processes and a representation of the model knowledge and that exhibits incremental behaviour. A model behaves incrementally if it produces output while reading input.

The cascaded architecture of incremental models brings with it a unidirectional information flow with no feedback, ie an incremental process does not send information to processes further up in the cascade. While this no-feedback condition keeps the model simple it is also a source of errors, because there is no way to inform a previous process about how far processing has proceeded or about the detection of an error. Instead, such information can only be obtained via a monitoring component that compares the output of the generation system – the output of the language comprehension system in the case of language production – and compares it to the planned output. This strong no-feedback condition can be softened without sacrificing the unidirectionality of the information flow by allowing indirect feedback. In this kind of feedback no explicit information is given back, but the effects of computations influence previous components, which can chiefly be used to inform about how processing has proceeded. Indirect feedback can be realised by letting two processes use a shared memory representation or increment buffers.

Incrementality can vary along a number of dimensions. The most important of these are monotonicity, buffering, lookahead, and feedback. Monotonic incremen-
tal models provide no possibility of correcting errors, while non-monotonic ones possess the means of update increments, which modify a previous increment. Buffering means that increments can be stored temporarily in increment buffers. This is required particularly in models in which it cannot be ascertained that an incremental process is ready for reading the next input increment when the previous process produces one. Incremental processes that have some lookahead it can operate more robust, because not only the current increment is available but also some 'future' ones. (Put the other way round it is not the most recent element that is processed, so processing in fact lags behind the number of elements it 'looks ahead'.) I developed \textsc{inC} as a model that uses no lookahead at all on the basis of what I called \textit{Extended Wundt's Principle}: Each processing component will be triggered into activity by a minimal amount of its characteristic input and produces characteristic output as soon as a minimal amount of output is available. This is an extension of Wundt's Principle as it was formulated by Levelt (1989:26).

However, incrementality is not only a way of performing conceptualisation in a particular way, it has also repercussions on the conceptualisation task. The most notable one is that preverbal messages are not generated as whole propositions but incrementally. I therefore also called them \textit{incremental preverbal messages}, which can be characterised as \textit{sequences of well-formed propositional structures on a sub-propositional level}. Another example is that perspectivisation need not be modelled as separate, transformational process, but as an effect of incrementality: each time a concept to be verbalised is selected the perspective of the ensuing verbalisation is also narrowed down further.

Incremental processing requires \textit{dynamic representations}, ie representations that can change over the course of time. Apart from the formalism \textit{referential nets} I used throughout this investigation I also demonstrated how the extendability property of semantic underspecification formalisms can be employed on the level of preverbal messages. In this way it becomes possible to connect the conceptualiser to an incrementally working formulator that encodes the incremental preverbal messages linguistically.

Incrementality has some similarities to \textit{anytime} processing. Both methods reduce the amount of the required resources and are ways to build models with \textit{bounded rationality}. It is even possible to construct incremental models that exhibit anytime behaviour. However, there are also significant differences. Whereas anytime processing is close to the notion of \textit{satisficing}, because it calculates the trade-off between the time spent on a computation and the gain in quality, incrementality is better described by the \textit{fast and frugal heuristics} approach to bounded rationality, because it relies on making quick decisions – based on cues retrieved from the available knowledge – that are reliable nevertheless.

There are several points that corroborate the \textit{cognitive adequacy} of the proposed methods in general and of \textsc{inC} in particular. Firstly, computing the focussed element in relation to the available knowledge in a local context is an efficient way of processing, suited to cope with the limitations of cognitive resources, and it models attentional mechanisms. Secondly, the piecemeal way of processing and the parallel
processing of an information stream on multiple stages, which includes the simultaneous reading of input and production of output, is an adequate way for a model to perform the given task, viz the online description of events. All of these conditions are fulfilled by humans performing the same task as well. Thirdly, Extended Wundt’s Principle is a means to cope with the fact that deliberation time is short. Thus, output is generated as soon as possible. Fourthly, the way of incrementality used in \( \text{INC} \) models the use of bounded rationality, ie no explicit optimisation or time–quality trade-offs are are carried out. This is also mainly due to Extended Wundt’s Principle. Finally, simulations showed that \( \text{INC} \) is a realistic model of the human conceptualiser.

For the kind of incrementality used for \( \text{INC} \) I provided a *formalisation* in the specification language Z. This specification can be taken and extended for building other incremental models that have the same overall architecture as \( \text{INC} \). These are in particular models that have a cascaded architecture, a shared memory representation, and use increments as informational units.

The methodology I used in this investigation is for the most part in the tradition of a *cognitive modelling* approach. However, in contrast to most research carried out in this field nowadays, I was for the most part not concerned with matching a set of empirical data onto a computational model but to develop a model that covers a comparatively large stretch of cognition. In order to do so I used AI and NLG methods. This way of doing NLG/AI research is not in the mainstream of these disciplines, which are not mainly concerned with cognitive considerations, either. Nevertheless, my approach can be regarded as ‘bionics of AI’, ie a field of research where results from observing nature is transferred to applications. Furthermore, the approach of building a model without having all empirical data in advance has the advantage that it does not run into the danger of giving *post hoc* explanations. Vice versa this means that \( \text{INC} \) should now be taken and compared to (more) empirical data.

In its current state of development \( \text{INC} \) models all tasks that are absolutely necessary for conceptualisation to be carried out. Nevertheless, it can be extended in many ways. I outlined how \( \text{INC} \), which is a model of the message generation part of conceptualisation, can be extended by a *monitoring* component. I also suggested a number of other possible extensions and how further research can be conducted on the basis of this investigation.

I used the formulation *thinking for speaking* by Slobin in many different forms. So, let me conclude by playing with this phrase a little. Since this investigation was mainly about the fact that thinking and speaking take place at the same time I was concerned with thinking *while* speaking. Next, the incremental mode of processing requires a certain kind of *representations* for speaking, in particular dynamic representations, ie representations that can change over time. I was interested only in one instance of thinking (conceptualisation), viz thinking for *speaking*, as opposed to, say thinking for *navigating*. Vice versa, I left out other important issues, eg grammatical encoding, which means that I investigated *thinking* for speaking.
SINCE not all readers will be familiar with the specification language Z, I will give a brief overview of those elements that are used in the specification in chapter 6. A complete reference of the version of Z that I use can be found in Spivey (1992).

Schemas. The main means to structure a Z specification are schemas:

By splitting the specification into schemas, we can present it piece by piece. Each piece can be linked with a commentary which explains informally the significance of the formal mathematics. In Z, schemas are used to describe both static and dynamic aspects of a system. The static aspects include: the states it can occupy; the invariant relationships that are maintained as the system moves from state to state. The dynamic aspects include: the operations that are possible; the relationship between their inputs and outputs; the changes of state that happen. (Spivey 1992: 2)

A schema can be written in two different but equivalent notations:

\[
\text{Schema} \begin{array}{l}
S_1 \\
\Delta S_2 \\
v : S_4 \\
pred_1 \\
pred_2
\end{array}
\]

A schema has a name (Schema in this case) and consists of two parts, which are separated by a dividing line. In the first part variables are declared, eg v in this schema, in the second part the relationships between the values of the variables are specified (Spivey 1992: 3). Schemas are also types, see below.
In the declaration part already defined schemas can be included. This is called
\textit{schema-reference}. So, if $S_1$ were defined as $S_1 \equiv [u : N]$ this schema is equivalent to
the expanded schema:

\[
\text{Schema} \equiv [u : N; \Delta S_2; v : S_4 | \text{pred}_1 \land \text{pred}_2]
\]

The notation $\Delta S_2$ for an already defined schema $S_2$ is used for operations on this
schema. This means, it introduces a primed variant for each defined variable. For
example, if $S_2$ is defined as $S_2 \equiv [w : N]$ then not only $w$ but also $w'$ is defined
in Schema. $w$ in this case contains the value before the operation, $w'$ the value
after the operation. Each pair of variables is implicitly constrained to satisfy the
invariant given by the schema predicate (the part below the line). Hence, it must
hold before as well as after the operation. (It is possible to speak of predicate as
well as of predicates. The single predicates in the first notation are connected via
conjunctions, as is made explicit in the second notation.)

The predicate part contains Z-expressions. Expressions have a truth value, most
of all set-relations and formulas of predicate logic. Note that $X \Rightarrow Y$ is the Z nota-
tion for the implication, while $X \rightarrow Y$ defines a total function, see below. Fur-
thermore, the if-then-else construction is allowed with its usual interpretation.

The following notational conventions can be used in referring to a schema:

\begin{itemize}
\item $S[a/b]$ is a schema-reference of $S$, but the variable $b$ is renamed to $a$
\item $S \setminus (a, b)$ is the definition of the schema without the variables $a$ and $b$
\item $S.\text{var}$ selects the variable $\text{var}$ from the schema $S$
\item $x?$ means that the $x$ contains an input value, $x!$ that it contains an output value
\item $X \equiv Y$ introduces an abbreviation, ie every occurrence of $X$ is short for $Y$
\end{itemize}

\textit{Mathematical expressions.} Mathematical expressions can be used at any place in
the specification in order to impose constraints, for example:

\begin{align*}
A \subseteq B \\
B \subseteq C \\
D &= E \cup A \\
A \setminus B &\neq \emptyset
\end{align*}

Quantified predicates are written in the form $Q \ S \bullet E$ where $Q$ is the existential
or the universal quantifier ($\exists$ or $\forall$), $S$ a schema text, ie a declaration (of variables)
and an optional list of predicates constraining the declaration, written $D \mid P$, and $E$
is the quantified expression. For example the predicate

\[
\forall x : N | 1 \leq x \leq 10 \bullet x \mod 2 = 1
\]

is true for all odd natural numbers between 1 and 10. Similarly the set $\{1, 3, 5, 7, 9\}$,
for whose elements this predicate is true, can either be enumerated in this fashion
or be written as:

\[
\{x : N | 1 \leq x \leq 10 \bullet x \mod 2 = 1\}
\]
Global variables can be introduced by axiomatic descriptions. For example
\[
\text{succ} : \mathbb{N} \rightarrow \mathbb{N} \\
\forall n : \mathbb{N} \cdot \text{succ}(n) = n + 1
\]
specifies the successor function on the natural numbers (\(\mathbb{N}\)). The (optional) part below the dividing line imposes constraints on the values of \(\text{succ}\).

Sets and Types. Basic sets are introduced as follows:

\[
\text{[BasicSet]}
\]
Further on the specification of such a set must be defined, e.g., by enumerating all elements. Apart from the usual relations defined on sets, like intersections or subsets, which I will not list here, the following notations are important in Z:

- \(\mathcal{P}X\) is the power-set of \(X\)
- \(\text{dom} R\) denotes the domain of relation \(R\)
- \(\text{ran} R\) denotes the range of relation \(R\)
- \(\mathbb{N}\) is the set of natural numbers (including 0)
- \(\mathbb{Z}\) is the set of integers

For making notations easier and more concise, a free type can be defined by enumerating its values:

\[
T ::= X_1 | X_2 | Y_1(\mathcal{T}) | Y_2(\mathcal{Z}_1, \mathcal{Z}_2)
\]
This makes it easy to specify recursive type definitions by repeating the type \(T\) in the definition. Furthermore, the following relations hold: \(X_1, X_2 : T\) (\(X_1\) and \(X_2\) are of type \(T\)), \(Y_1 : T \rightarrow T\), and \(Y_2 : \mathcal{Z}_1 \times \mathcal{Z}_2 \rightarrow \mathcal{T}\), and the sets \(X_1, X_2, \text{ran } Y_1, \text{ and ran } Y_2\) are disjoint and partition \(T\). (\(X \rightarrow Y\) is an injection from \(X\) to \(Y\), see below.)

For example a binary tree containing natural numbers can be defined as (Spivey 1992: 83f):

\[
\text{TREE} ::= \text{tip} | \text{fork}(\mathbb{N} \times \text{TREE} \times \text{TREE})
\]
Since free type definitions add nothing to the expressive power of Z, there are always equivalent definitions.

Relations. \(X \times Y\) is the Cartesian product of the two sets \(X\) and \(Y\). The \(X \leftrightarrow Y\) notation defines the set of binary relation between \(X\) and \(Y\). This is equivalent to \(\mathcal{P}(X \times Y)\). The more specific relations are written as:
Sequences. Sequences are a predefined in Z. ‘seq X is the set of finite sequences over X. These are finite functions from N to X whose domain is a segment 1 . . n for some natural number n.’ (Spivey 1992: 115) The following notations for sequences are used in chapter 6:

- ⟨i₁, . . . , iₙ⟩ gives the sequence explicitly
- ⟨⟩ is the empty sequence
- sq₁ ++ sq₂ concatenates the sequences sq₁ and sq₂
- tail and head serve to decompose a sequence; tail returns the sequence without the first element; head returns the first element
- sq₁ in sq₂ means that sq₁ is a sub-sequence of sq₂, eg ⟨b, c⟩ in ⟨a, b, c, d⟩ is true
- #sq is the length of sequence sq

Generic constants. Schemas with a double top line introduce generic constants. These are ‘mathematical constructions that are independent of the elements from which the construction starts’ (Spivey 1992: 38). The notions of reflexivity and symmetry can, for example, be defined as follows:

\[
\begin{align*}
\text{[X]} \\
\text{reflexive}_\sim : P(X \leftrightarrow X) \\
\text{symmetric}_\sim : P(X \leftrightarrow X)
\end{align*}
\]

\[
\forall R : X \leftrightarrow X \bullet \\
(\text{reflexive } R \iff \text{id } X \subseteq R) \land \\
(\text{symmetric } R \iff R = R^\sim)
\]

id X is the identity relation, R^\sim the inverse relation. The ‘\sim’ indicates where the arguments of the defined relations are written. Here, both relations are prefix. After this definition has been given a relation A can be said to be reflexive, for example, by saying reflexive A.
THE EXAMPLE discussed in chapter 14 is based on an actual simulation carried out with the implementation of inC. The system output is given in this appendix. Please note that the numbering of refOs differs from the one in chapter 14, because the system assigns the numbers whereas in the discussion I chose a different numbering for presentational reasons. The activation of designations is written as <<...>>, and the refOs are written as simple numbers, i.e., without leading r (except in the headline of a refO).

B.1 The input coming from the PPU

PEs are generated by the PPU as a sequence of lines of text. The first line consists of the number of the PE with which it can be unequivocally be identified by the PPU as well as inC. The second line states whether it is a new PE or an update for an already existing one. In the former case the following lines give the attributes (third line) and the designations of the refO with which the PE will be represented (fourth line). In the latter case these lines contain the attributes to be removed (third line), the attributes to be added (fourth line), the designations to be removed (fifth line), and the designations to be added (sixth line).

1
new
object
plane
eta x: plane(x)
2
new
situation
at_time(0 200)
nil
3
new
path
nil
eta x: chpos(#2# #1# x); eta x: straight(x)
2
update
nil
nil
nil
eta x: chpos(x #1# #3#)
1
update
nil
nil
nil
eta x: chpos(#2# x #3#)
4
new
ref_sys_spat
spatial_entity
iota x: R_sys(x #3#)
5
new
region
nil
iota x: Left_region(x #4#)
6
new
region
nil
iota x: Right_region(x #4#)
7
new
situation
at_time(201 320);met_by(#2#)
nil
2
update
nil
meets(#7#)
il
nil
8
new
path
nil
eta x: chpos(#7# #1# x); eta x: curved(x); eta x to(x #5#)
7
update
nil
nil
nil
eta x: chpos(x #1# #8#)
1
update
nil
nil

250
B.2 Examples of generated incremental preverbal messages

This is the PVM file of inC version 0.2.2 (build 077)
DoAt=0.5; LoTB=6; LT=6050; AT=0.5
Domain: motionEvents
Selection Strategy: mostComplex
Real-Time: true; Time Frame: 50

New PVM starting with ref0 16

situation -------- r18 ---- eta x: chpos(x 19 20) <<0.5>>
verb_of(16)
parts([8 13])
at_time(0 320)
status(regular)

object ----------- r19 ---- eta x: plane(x) <<0.9>>
verb_of(7)
plane
status(regular)
pe(1)

path ------------- r20 ---- eta x: to(x 21) <<0.5>>
verb_of(17) eta x: p_curved(x) <<0.6>>
parts([9 14])
status(regular)

region --------- r21 ---- iota x: Left_region(x 22) <<0.5>>
verb_of(11)
status(regular)
pe(5)

ref_sys_spat -------- r22 ----
verb_of(10)
spatial_entity
status(regular)
pe(4)

This is the PVM file of inC version 0.2.2 (build 077)
DoAT=0.5; LoTB=6; LT=125; AT=0.4
Domain: motionEvents
Selection Strategy: mostComplex
Real-Time: true; Time Frame: 50

New PVM starting with refO 8

situation --------- r13 ---- eta x: chpos(x 14 15) <<0.5>>
verb_of(8)
at_time(0 200)
status(regular)
pe(2)

object -------------- r14 ---- eta x: plane(x) <<0.9>>
verb_of(7)
plane
status(regular)
pe(1)

path ---------------- r15 ---- eta x: straight(x) <<0.6>>
verb_of(9)
status(regular)
pe(3)

New PVM starting with refO 19

situation --------- r21 ---- eta x: chpos(x 22 23) <<0.5>>
verb_of(19)
parts([8 16])
at_time(0 320)
status(regular)

object -------------- r22 ----
verb_of(7)
plane
status(regular)
pe(1)

path ---------------- r23 ---- eta x: transitionpoint(24 x) <<0.4>>

252
This is the PVM file of inC version 0.2.2 (build 077)
DoAT=0.5; LoTB=6; LT=125; AT=0.6
Domain: motionEvents
Selection Strategy: retainGranularity
Real-Time: true; Time Frame: 50

New PVM starting with refO 0

situation ----------- r13 ---- eta x: chpos(x 14 15) <<0.5>>
verb_of(8)
at_time(0 200)
status(regular)
pe(2)

object ------------- r14 ---- eta x: plane(x) <<0.9>>
verb_of(7)
plane
status(regular)
pe(1)

path ------------- r15 ---- eta x: straight(x) <<0.6>>
verb_of(9)
status(regular)
pe(3)

New PVM starting with refO 16

situation ----------- r21 ---- eta x: chpos(x 22 23) <<0.5>>
verb_of(16)
part_of([19])
met_by(8)
at_time(201 320)
status(regular)
pe(7)

object ------------- r22 ----
verb_of(7)
plane
status(regular)
pe(1)
This is the PVM file of inC version 0.2.2 (build 077)
DoAT=0.5; LoTB=6; LT=125; AT=0.5
Domain: motionEvents
Selection Strategy: retainGranularity
Real-Time: true; Time Frame: 50

New PVM starting with refO 8

situation ----------- r13 ---- eta x: chpos(x 14 15) <<0.5>>
verb_of(8)
at_time(0 200)
status(regular)
pe(2)

object -------------- r14 ---- eta x: plane(x) <<0.9>>
verb_of(7)
plane
status(regular)
pe(1)

path ---------------- r15 ---- eta x: straight(x) <<0.6>>
verb_of(9)
status(regular)
pe(3)

New PVM starting with refO 16

situation ----------- r21 ---- eta x: chpos(x 22 23) <<0.5>>
verb_of(16)
part_of([19])
met_by(8)
at_time(201 320)
status(regular)
pe(7)

object -------------- r22 ----
verb_of(7)
plane
status(regular)
pe(1)

path ---------------- r23 ---- eta x: to(x 24) <<0.5>>
verb_of(17)
et x: curved(x) <<0.6>>
part_of([20])
status(regular)
pe(8)

region -------------- r24 ---- iota x: Left_region(x 25)  
verb_of(11)
status(regular)
pe(5)

ref_sys_spat ------ r25 ----
verb_of(10)
spatial_entity
status(regular)
pe(4)

===================================================
This is the PVM file of inC version 0.2.2 (build 077)
DoAT=0.5; LoTB=6; LT=125; AT=0.1
Domain: motionEvents
Selection Strategy: retainGranularity
Real-Time: true; Time Frame: 50

===================================================
New PVM starting with refO 8

situation ----------- r13 ---- eta x: chpos(x 14 15)  
verb_of(8)
at_time(0 200)
status(regular)
pe(2)

object -------------- r14 ---- eta x: plane(x)  
verb_of(7)
plane
status(regular)
pe(1)

path ---------------- r15 ---- eta x: straight(x)  
verb_of(9)
status(regular)
pe(3)

===================================================
New PVM starting with refO 16

situation ----------- r21 ---- eta x: chpos(x 22 23)  
verb_of(16)
part_of([19])
met_by(8)
at_time(201 320)
status(regular)
pe(7)

object -------------- r22 ----

255
B.3 A final state of the ccr

object --------------- r7 ---- eta x: chpos(19 x 20) <<0.5>>
plane eta x: chpos(16 x 17) <<0.5>>
status(regular) eta x: chpos(8 x 9) <<0.5>>
pe(1) eta x: plane(x) <<0.9>>

situation ------------ r8 ---- eta x: chpos(x 7 9) <<0.5>>
part_of([19])
meets(16)
at_time(0 200)
status(regular)
pe(2)
path --------------- r9 ----- eta x: finalpoint(18 x) <<0.1>>
part_of([20]) eta x: straight(x) <<0.6>>
status(regular) eta x: chpos(8 7 x) <<0.5>>
pe(3)

ref_sys_spat ------- r10 ---- R_sys(x 9) <<0.5>>
spatial_entity status(regular)
pe(4)

region --------------- r11 ---- Left_region(x 10) <<0.5>>
spatial_entity status(regular)
pe(5)

region --------------- r12 ---- Right_region(x 10) <<0.5>>
spatial_entity status(regular)
pe(6)

situation ------------- r13 ---- eta x: chpos(x 14 15) <<0.5>>
verb_of(8) at_time(0 200)
status(regular)
pe(2)

object --------------- r14 ---- eta x: plane(x) <<0.9>>
verb_of(7) plane status(regular)
pe(1)

path --------------- r15 ---- eta x: straight(x) <<0.6>>
verb_of(9) status(regular)
pe(3)

situation ------------- r16 ---- eta x: chpos(x 7 17) <<0.5>>
verb_of([19]) met_by(8)
at_time(201 320)
status(regular)
pe(7)

path --------------- r17 ---- eta x: startpoint(18 x) <<0.1>>
part_of([20]) eta x: to(x 11) <<0.5>>
status(regular) eta x: curved(x) <<0.6>>
pe(8) eta x: chpos(16 7 x) <<0.5>>

location -------------- r18 ---- eta x: transitionpoint(x 20) <<0.4>>
status(regular) eta x: startpoint(x 17) <<0.1>>
pe(9) eta x: finalpoint(x 9) <<0.1>>
situation --------- r19 ---- eta x: chpos(x 7 20) <<0.5>>
parts([8 16])
at_time(0 320)
status(regular)

path -------------- r20 ---- eta x: transitionpoint(18 x) <<0.4>>
concat([9 17])
status(regular)
eta x: curved(x) <<0.6>>
eta x: chpos(19 7 x) <<0.5>>

situation --------- r21 ---- eta x: chpos(x 22 23) <<0.5>>
verb_of(16)
part_of([19])
met_by(8)
at_time(201 320)
status(regular)
pe(7)

object ----------- r22 ----
verb_of(7)
plane
status(regular)
pe(1)

path -------------- r23 ---- eta x: curved(x) <<0.6>>
verb_of(17)
part_of([20])
status(regular)
pe(8)

B.4 The DaVinci output window of the ccr

The window shown below is the output of the ccr that is generated with the help of the DaVinci graph drawing tool. Each refO is represented by a node, relations by edges. The different kinds of refOs and relations are kept apart by a colour scheme. In this visualisation the chpos relation is not given, because this leads to very complex graphs. The startpoint and finalpoint relations are not given too. Accordingly, the given graph consists of non-connected subgraphs. The leftmost subgraph contains the situation refOs with the numbers 8, 16, and 19. The refOs with the numbers 13 and 21 are the two verbalisations that were generated for the two simpler situations. The next subgraph contains the refO representing the plane (7) and its two verbalisation refOs. The rightmost subgraph represents the spatial entities: 9, 17, 20 are the three paths, 18 is the location where the simpler paths meet, 10 is the spatial reference system, and the 11 and 12 and the left and right region, respectively. 15 and 23 are the verbalisation refOs of the simpler paths.
B.4 THE DAVINCI OUTPUT WINDOW OF THE CCR
VERBALISATIONS OF MOTION EVENTS

The verbalisations I provide in this chapter were recorded in an ongoing verbalisation study, i.e. the data are not yet fully recorded and evaluated. What follows are verbalisations of the first five participants. The scenes that were presented to elicit the verbalisations given here are the ones from the study that are similar to the scene discussed in section 13.3. The first scene differs in that both planes come into view simultaneously, the second scene in that the first plane, i.e. the one turning off, moves on Runway 1, i.e. on the upper runway, and the other plane on Runway 2. Words in small caps were uttered with an emphasis.

Participant 1, scene 1. Jetzt kommt zwei Flugzeuge von links. Aah... herein und, ähm, auf Laufbahn eins und zwei, also parallel und dann, äh, ist das untere Flugzeug, das auf... äh... Laufbahn zwei war... nach links abgebogen und hat dann gewartet bis das.. Flugzeug auf der.. ersten Laufbahn an ihm vorbeigefahren is

Participant 1, scene 2. Jetzt kommt von links ein Flugzeug, äh, reingefahren... macht eine rechtsbiegung, auf der ersten Rollbahn war das, jetzt kommt par, äh, gleichzeitig ein... äh, Flugzeug auf der zweiten Rollbahn von links, so m.deswegen muss das erste Flugzeug halten... und das, äh, zweite Flugzeug ist jetzt an dem... haltenden Flugzeug vorbeigefahren und fährt gradeaus weiter auf der zweiten Rollbahn

Participant 2, scene 1. Jetzt kommen zwei, eins auf der ersten Rollbahn, 's andere is auf der zweiten, 's auf der zweiten biegt an der ersten Abzweigung... will also auf die erste, bleibt kurz stehn, läset das andere vorbeifahen, das immer gerade weiterfährt

Participant 2, scene 2. 'n Flugzeug kommt von links auf Rollfeld eins... biegt nach rechts ab, 'n Flugzeug kommt von links auf Rollfeld zwei... das erste Flugzeug läset das auf Rollfeld 2 vorbeifahren... und das fährt gradeaus weiter...

Participant 3, scene 1. auf dem Rollweg zwei kommt ein schnelles Flugzeug gefahren, auf Rollweg 1 ein... laangsames, das auf Rollweg 2 biegt nach links ab in die
erste Abzweigung, das auf Rollweg 1 fährt weider, das auff .. eben Abgebogene hält an .. mm .. um das andere passieren zu lassen

*Participant 3, scene 2.* Æh, auf dem Rollweg 1 kommt von links das nächste Flugzeug, biecht gleich nach rechts auf, Æh, die Abzweigung rauf, auf, Æh, Rollweg 2 kommt wàhrenddessen auch ein Flugzeug .. das von Rollweg 1 abgebogene Flugzeug hält an, um das auf Rollweg 2 fahrende Flugzeug passieren zu lassen .. das Roo .. auf Rollweg 2 pf .. fahrende Flugzeug fährt immer gradeaus, das, Æhm, abgebogene Flugzeug von Rollweg 1 hält seine Position,

*Participant 4, scene 1.* Æs gradeaus übers Rollfeld, links .. erstes Rollfeld und zweites Rollfeld .. obere, oben startet einer, fährt gradeaus, linke .. zweite Rollbahn .. is einer gestartet, links abgebogen und biecht jetzt auf die .. erste Rollbahn ein und fährt .. hinter dem .. annern Flugzeug her, was sich auf der ersten Rollbahn befindet, weil .. das ff, erste verschwindet, das zweite, was dort eingebogen is verschwindet jetzt auch .. gleich .. rechts ..

*Participant 4, scene 2.* von links in Richtung rechts auf Rollfeld 1 .. erste Abzweigung rechts, von links auf Rollfeld zwei kommt ein zweites flugzeug, das erste wartet .. an Roll, an der Abzweigung Rollfeld 2, das zweite bleibt immer noch auf Rollfeld 2, fährt gradeaus in Richtung .. rechter Bildrandt ..

*Participant 5, scene 1.* Æhm, zwei Flugzeuge bewegen sich jeweils auf Runway eins und zwei .. von Runway zwei biegt nach links ab und bewegt sich auf Runway 1 zu, stoppt .. bevor es, Æhm, das Flugzeug auf Runway 1 trifft

*Participant 5, scene 2.* ein Flugzeug kommt von der linken Seite auf Rollbahn eins, biegt rechts ab .. und bewegt sich auf Rollbahn 2 zu, wo grade ein weiteres Flugzeugk, Æhm, von links nach rechts fährt, sch .. das erste Flugzeug stoppt .. lässt das zweite Flugzeug passieren auf Rollbahn 2 .. das Flugzeug fährt weiter.
adaptation. Adaptation is the adjustment of the behaviour of a computing device in a way that its available (allocated) resources suffice to accomplish a task.

agent. There is no generally accepted definition of what is an agent. However, an agent has at least the following three properties: (1) it acquires knowledge by perceiving its environment, (2) it processes the perceived knowledge so as find a way to attain a goal, (3) it executes actions to influence its environment so as to get closer to its goal or reach it. An agent typically has (most of) the following properties: (1) reactivity, (2) autonomy, (3) collaborative behaviour, (4) communication ability, (5) inferential capability, (6) temporal continuity, (7) personality, (8) adaptivity, (9) mobility. Different types of agents can be distinguished: (1) reactive (behaving according to stimulus–response pairs), (2) deliberative (agent has plans, goals, intentions), (3) situated (located in a dynamic environment), (4) autonomous (self-starting behaviour, often robots), (5) rational (reflecting and evaluating own actions), (6) social (taking social goals into account), [Strube1996] [Bradshaw1997].

algorithm. There is no general agreement on what an constitutes an algorithm. Usually the following properties are mentioned: (1) parameters, input (to specify a problem from the problem class described by the algorithm), (2) output, (3) determinate (same output for same input), (4) finite length (an algorithm is described by a finite number of instructions), (5) finiteness (an algorithm terminates after a finite number of steps), (6) determinism, definiteness (there is always a unique next step), (7) effectiveness. Some of these properties may be missing, eg in the context of operating systems the property that an algorithm must terminate is usually dropped, because an operating system should run infinitely [Knuth1973] [Bibliographisches Institut1986].

allocation. The allocation of a resource is the assignment of (part of) the resource to a task in order for the task to be carried out.

anytime algorithm. An anytime algorithm is an algorithm that refines its res-
ult over time so that they can produce a result at any time. The later the result is produced the more accurate their response is. One major variant of an anytime algorithm is that it explicitly estimates the benefit of investing a certain amount of resources, especially time. Zilberstein (1996) gives the following seven properties of anytime algorithms: ‘First is measurable quality: The quality of an approximate result can be determined precisely. […] Second is recognizable quality: The quality of an approximate result can easily be determined at run time (that is, within a constant time). […] Third is monotonicity: The quality of the result is a nondecreasing function of time and input quality. Note that when quality is recognizable, the anytime algorithm can guarantee monotonicity by simply returning the best result generated so far rather than the last generated result. Fourth is consistency: The quality of the result is correlated with computation time and input quality. […] Fifth is diminishing returns: The improvement in solution quality is larger at the early stages of the computation, and it diminishes over time. Sixth is interruptibility: The algorithm can be stopped at any time and provide some answer. […] Seventh is preemptability: The algorithm can be suspended and resumed with minimal overhead.’ The quality of an output is determined by a metric, three of which are: ‘First is certainty, a measure of the degree of certainty that the result is correct. The degree of certainty can be expressed using probabilities, fuzzy set membership, or any other approach. Second is accuracy, a measure of the degree of accuracy, or how close the approximate result is to the exact answer. Typically, with such algorithms, high quality guarantees that the error is below a certain upper bound. Third is specificity, a metric of the level of detail of the result. In this case, the anytime algorithm always produces correct results, but the level of detail is increased over time.’

articulator. The component of the Levelt (1989) model that computes and controls articulatory movements from the phonetic plan.

AT (activation threshold). The activation threshold serves in determining whether an element (a refO or a designation in INC) is verbalised. If the activation of a designation is below AT then it is only used in the verbalisation if it is semantically required by the preverbal message.

available knowledge. The available knowledge is the model knowledge that is accessible by an incremental process and its incremental algorithms.

— B —

best match. The concept in the CS that has the highest degree of agreement (DOA) with the candidate list.

blackboard. A common memory location for AI systems. It is used to hold intermediate results. There are interdependent modules that will post data to the blackboard and use data from the blackboard in new computations. These modules are sometimes referred to as a group of collaborating experts. The blackboard may con-
sist of two parts: the domain blackboard (reasoning about the domain) and the scheduling blackboard (reasoning about scheduling) (Mercadal 1990: 31).

**candidate list.** List of entities (refOs) from the ccr that is handed over from construction to the concept matcher. The concept matcher tries to match the elements of this list onto the concepts stored in the cs.

cascade. A cascade consists of incremental processes that work in parallel and that are arranged in a fixed, sequential order in such a way that each process has one preceding and one succeeding process. The incremental processes of a cascade are connected by increment streams and/or increment buffers. A process reads input increments from its preceding process and sends output increments to its succeeding process. The first process of a cascade has no preceding process but reads input from the environment, and the last process has no succeeding process but sends its output to the environment.

category. A category is a set of abstract or concrete entities that share a set of common properties. Their mental representations are called concepts.

**ccr (current conceptual representation).** The internal representation of external states of affairs. It is a referential net (Habel 1986). It is a shared memory that is built up by construction and can be accessed by selection, linearisation, and pvm-generation. Apart from pes it contains refOs that were computed by the concept matcher and the list of verbalisation refOs (in the traverse).

**component.** A component is a part of a model or system that is independent of other parts. The extreme form of independence of a component is a module. In difference to a module the informational encapsulation need not be absolute. The components of the conceptualiser are modelled by processes in inc.

**concept.** A concept is the mental representation of a category or an entity. They are the symbolic elements from which a conceptual representation is constructed.

**concept matcher.** The process that matches the candidate list received from construction onto the entries in the cs. It determines the best match, the match list, and the degree of agreement (doa), which it returns to construction.

**concurrency.** Concurrency is simulated parallelism. It is used for simulating the execution of multiple processes – then often called tasks – on one processor.

**construction.** The construction process takes pes it receives from the ppu and builds up the hierarchical representation of the ccr from these pes in cooperation with the concept matcher.
A referential net \cite{Habel1986}, which contains rules on how complex concepts can be constructed from simpler ones. The hierarchical structure of an entry in the cs can be regarded as a production: the simpler concepts are the guards, building up the more complex concept is the action.

**current preverbal message.** The current preverbal message is the incremental preverbal message that is currently produced by pvM-generation. The increments contained in the current preverbal message are accessible by pvM-generation so that it can (1) keep track of still missing parts of the incremental preverbal message and (2) compute extensions, modifications, and self-corrections.

**direction.** A spatial entity that in standard mathematical terminology represents the orientation or sense of an entity. Here, I use the more common word direction instead.

**DOA (degree of agreement).** The degree of agreement is determined by the concept matcher. It is a measure of the quality of the match between the candidate list and an entry in the cs. DOA is a value between 0 and 1.

**DOAT (degree of agreement threshold).** The degree of agreement threshold is one of INC’s parameters. It is used by construction to decide whether a best match is good enough to be inserted into the ccr. If DOA \( \geq \) DOAT then the best match is inserted. If DOA < DOAT the best match is ignored. DOAT is a value between 0 and 1.

**event.** An event is a situation type that consists of a change and (at least) one culmination point. In this investigation it is used synonymously to situation.

**event thread.** An event thread is a sequence of events that have the same bearer.

**Extended Wundt’s Principle.** Each processing component will be triggered into activity by a minimal amount of its characteristic input and produces characteristic output as soon as a minimal amount of output is available.

**focussed element.** A focussed element is an element of the representation of the available knowledge. Each incremental process has one focussed element. It is the entry point to the available knowledge of an incremental algorithm executed by the incremental process. An incremental algorithm starts evaluating the available knowledge from this element.

**formulator.** The formulator encodes a preverbal message linguistically and generates a phonetic plan.

**frame problem.** The frame problem was first stated by \cite{McCarthyHayes1969}. I use it synonymously with the persistence problem, the problem to decide which

---

\( cs \) (concept storage).
facts (knowledge, part of the representation) are changed by an action and which are unaffected (persist).

**head of traverse buffer.** The head of the **traverse buffer** is the first element in the **traverse buffer**. This element is the candidate for the next element that is verbalised. It is the only element of the **traverse buffer** seen by **PVM-generation**. When the LT of this element has expired **PVM-generation** removes it from the **traverse buffer** and commences a **new incremental preverbal message**.

**increment.** An increment is a piece of information that is the input of an **incremental process** (**input increment**), the output of one (**output increment**), or both.

**increment buffer.** An increment buffer is a buffer between two or more **incremental processes**. It can store a limited number of increments that cannot be processed further at the moment, because the reading process is not yet ready to take an **increment** from the buffer as input **increment**.

**increment stream.** An increment stream is an ordered sequence of **increments** that changes over time. **Increments** can be appended to and popped from the increment stream.

**incremental algorithm.** An incremental algorithm is an **algorithm** that has the following properties: (a) it obtains a triple of **focussed element**, **local context**, and **available knowledge** as input; (b) it only considers the **local context** for its computations starting from the **focussed element**; (c) the output consists of modifications in the **available knowledge**, a **new focussed element**, and an **output increment**.

**incremental behaviour.** The **minimal** condition for calling the behaviour of a **model** incremental is that it is capable of producing output before it received all input possibly relevant for the correct and complete computation of the corresponding output. The **strong** condition for incremental behaviour is that additionally input and output are read/written in parallel.

**incremental model.** An incremental model is a **model** that contains a **cascade of incremental processes** and a representation of the **model knowledge**. The incremental model manages (a) the views of the **incremental processes** on the **model knowledge** and (b) the access of the **incremental processes** to the **model knowledge** so that only one **process** has access to the representation at a given point in time.

**incremental preverbal message.** An incremental preverbal message is a **preverbal message** in [Levelt 1989] sense with the additional emphasis on the incremental way of production. Therefore, it is a **sequence of well-formed propositional structures**
on a sub-propositional level. This refinement is necessary, because the standard view does not take into account that a proposition is too big an information structure to be an increment between conceptualiser and formulator. For this reason, the increments must be sub-propositional. Nevertheless, increments themselves can be represented by propositions. Thus, a preverbal message is a sequence of propositional structures that can be combined to one proposition, which, in turn, corresponds to the classic view on preverbal structures.

incremental process. An incremental process is a process that behaves incrementally. It reads input from an increment stream, writes its output to another increment stream, and recursively executes the following two steps: (1) it determines a local context and (2) calls an incremental algorithm with respect to the available knowledge. The recursion ends if (a) a new input increment is available or (b) the result(s) cannot be improved. In the second case the process suspends itself until a new input increment is available. An incremental process runs in an infinite loop until it is explicitly terminated from the outside.

incrementality. Incrementality is the property of a model, system, algorithm, or process to compute information in a piecemeal manner.

indirect feedback. Indirect feedback is feedback that is not realised as direct transmission of information. Instead, the component giving feedback alters a representation that the component receiving feedback is using as well. If the modification of the representation by the one component affects the operations of the other component an indirect feedback has been given.

information. Information is the difference between available knowledge and transferred knowledge.

internal speech. see Phonetic Plan.

— K — knowledge. The facts and rules of a model or system are its knowledge. The two main types of knowledge are declarative and procedural knowledge. Declarative knowledge is knowledge about facts (knowledge what), while procedural knowledge is knowledge about rules (knowledge how). The latter kind is mostly meant in this investigation when using the terms knowledge or knowledge representation.

— L — linearisation. The linearisation process can reorder the elements of the traverse buffer. Its goal is to bring these elements into an appropriate order with respect to the current verbalisation goals. The time it has available for performing this task is the IT, ie the time after an element is inserted into the traverse buffer by selection and before it is removed by PVM-generation.

local context. The local context is that part (subset) of the available knowledge
that is considered by an incremental algorithm in its computation. It is a connected sub-part of elements in the available knowledge around the focussed element. In each recursion of the algorithm the local context is different. It is determined by a heuristic that is specific to the incremental algorithm for whose call it is generated.

**location**. A location is a spatial entity that represents a position of an object in space.

**LOTB (length of traverse buffer)**. The length of traverse buffer is one of inC’s parameters. Its value defines the number of elements that can be stored in the traverse buffer.

**LT (latency)**. The latency is one of inC’s parameters. Its value specifies the span of time that selected nodes stay in the traverse buffer before they can be fetched by pvm-generation.

**match list**. The list returned by the concept matcher to the construction process that contains pairs as elements. Each pair assigns an element in the cs to an actual element in the ccr, if the ccr contains such an element.

**metric**. A metric determines the degree of usefulness of a resource for performing a task.

**model**. A model is a textual and/or formal description of tasks and representations. Its implementation is called a system.

**model knowledge**. The model knowledge is the declarative knowledge of an incremental model.

**module**. A module is a part of a model or system that is independent of other parts of the model or system. According to Fodor [1983] it must have the following nine properties: (1) domain specificity, (2) mandatoriness, (3) limited central access to the mental representations, (4) fast speed, (5) informational encapsulation, (6) ‘shallow’ outputs, (7) association with fixed neural architecture, (8) characteristic and specific breakdown patterns, (9) the ontogeny exhibits a characteristic pace and sequencing. A module has only some but not all of these properties is called a component.

**monitor**. The monitor is the component that reads in the parsed speech from the speech comprehension system and compares it with the planned preverbal message. If it detects a deviation it can generate a correction.

**monitored traverse**. The monitored traverse is the part of the ccr that contains the
information that has been received by the monitor from the language comprehension system. This information is correlated with the information in the ccr and the traverse. In the ideal case this is a one-to-one correspondence of refO.

- o – object. An object is a concrete real world entity.

- p – parallelism. Parallelism is a mode of computation in which the overall process is subdivided into multiple sub-processes and in which these sub-processes are carried out on different processors simultaneously.

path. A path is a spatial entity that represents the trajectory along which an object is moving or was moving.

pe (perceived entity). Perceived entities are results of the pre-processing and input to INC, where they are taken by the construction process. They are perceptually unanalysed – although they may be under different circumstances – and serve as the interface between perception and conceptualisation.

phonetic plan. Also internal speech. In Levelt’s (1989) model the interface between formulator and articulator. It contains output of the formulator and input to the articulator.

ppu (pre-processing unit). The perceptual component (a module in the implemented system) that takes the perpetual, continuous perceptual input stream and segments it into pes.

pre-processing. The computations necessary to get from different kinds of input to pes.

PVM-generation (preverbal-message generation). PVM-generation is the process that polls the head of traverse buffer. When the LT of the head of traverse buffer is over it starts generating an incremental preverbal message, which it hands over (incrementally) to the subsequent process, which is the first process of the formulator.

preverbal message. In Levelt’s (1989) model preverbal messages are propositions that constitute the interface between conceptualiser and formulator. They correspond to semantic representations. In the context of INC the notions incremental preverbal message and current preverbal message are particularly important.

process. A process, also called thread in the context of concurrency, is a sequence of states. It is a sequential program usually carried out in an infinite loop. A process executes a task that is part of the overall task of the system.
processor. A processor is a device that executes one or more processes.

resource. A resource is a – material or immaterial – auxiliary means required by a task in order to perform a function in achieving a goal. The performance or even the overall success of the task depends on allocation of the required resources, or, how much of the required resources are allocated to the task.

selection. The selection process selects refOs for verbalisation. Its local context is the traverse buffer, but it has access the ccr and the traverse as well. It can change the traverse buffer by two operations: append and replace. There is no simple deletion operation. Two selection strategies can be applied to each input increment from construction. The first strategy selects the most complex concept that is not yet selected for verbalisation. The second strategy attempts to retain the level of granularity of the selected concepts, ie the concepts currently in the traverse buffer. It only changes the granularity level if no refOs are in the traverse buffer.

shared memory. A shared memory is a memory that accessible by multiple processes. Particularly important is the fact that the processes do not partition the shared memory, ie there are no disjoint parts of the shared memory that each are accessed by only one process. Shared memory does not consist of messages that are passed from one process to another one but of an integrated knowledge representation.

dependent. In this investigation a situation is an abstract concept representing a time interval or a time point that relates an object and a spatial entity.

spatial entity. A spatial entity is an abstract concept that represents spatial knowledge. In this work it is one of path, location, or direction.

state. A state is the set of variables with their assigned values of a system at a given point of time. In particular, it is the content of a knowledge representation at a given point of time.

symbol grounding problem. The symbol grounding problem is the problem that symbols – concepts in particular – cannot be defined exclusively by other concepts, because this results in circular explanations. In order to avoid the circularity concepts must be expressed in non-conceptual terms.

system. A system is an implemented model. In order to build an implementation (an executable program) additional assumptions have to be made, while other issues can only be realised in a very reduced version due to complexity.

thread. See process. The term thread is used especially in contexts in which...
The path through the ccr consisting of the verbalisation refOzs that were generated by pvm-generation during the production of incremental preverbal messages.

The traverse buffer is an increment buffer that contains pointers to refOzs in the ccr. These refOzs were selected for verbalisation but are not yet verbalised. The pointers in the traverse buffer can be manipulated by the processes selection and linearisation until pvm-generation takes out the head of traverse buffer in order to start a new incremental preverbal message. If selection appends an element to the traverse buffer when it is filled, the head of traverse buffer is dropped.

-- u -- update increment. An update increment is an increment that updates an increment previously read or sent by an incremental process.

-- w -- Wundt's Principle. Each processing component will be triggered into activity by a minimal amount of its characteristic input.
BIBLIOGRAPHY


Foth, Kilian, Ingo Schröder, & Wolfgang Menzel, 2000b, A transformation-based

— G —


Guhe, Markus, in print, Incremental preverbal messages. In: *Härtl & Tappe [in print]*.


pp 123–128, Bamberg.

——, in print, Incremental generation of interconnected preverbal messages. In: Thomas Pechmann & Christopher Habel (eds), Multidisciplinary approaches to language production, Berlin: De Gruyter.


Härtl, Holden & Heike Tappe (eds), in print, Mediating between concepts and language – Processing structures. Berlin: De Gruyter.

Hartsuiker, Robert J & Herman HJ Kolk, 2001, Error monitoring in speech produc-


Kempen, Gerard, forthcoming, Grammatical Performance in Human Sentence Production and Comprehension. Experimental Psychology Unit, Department of Psychology, Leiden University.

———, in print, Word order scrambling as a consequence of incremental sentence production. In: (Härtl & Tappe in print).


Landau, Barbara & Ray Jackendoff, 1993, “What” and “where” in spatial language

279


[Pragmatik, Tübingen: Max Niemeyer Verlag, 1990].


Nuyts, Jan & Eric Pederson (eds), 1997, Language and Conceptualization. Cambridge:
Cambridge University Press.

— P —

— R —
Rosch, Eleanor, Carolyn B Mervis, Wayne D Gray, David M Johnson, & Penny Boyes-Braem, 1976, Basic Objects and Natural Categories. *Cognitive Psychology*
BIBLIOGRAPHY

8: 382–439.


---


---


BIBLIOGRAPHY

Press. Edited by John B Carroll. [Sprache, Denken, Wirklichkeit, Reinbeck: Rowohlt 1984].


———, in print, Semantics as a gateway to language. In: [Härtl & Tappe in print].


285
### NAME INDEX

| A | Adiwidjaja, Andy 46 |
|   | Aguilar, Cynthia 59, 60 |
|   | Allen, James F 68 |
|   | Altman, Gerry TM 55, 83, 85 |
|   | Anderson, John R 110, 148 |
|   | Aristotle 2 |
|   | Austin, John L 37 |
|   | Avrahami, Judith 8, 53 |
| B | Bach, Emmon 8 |
|   | Baddeley, Alan 38 |
|   | Barsalou, Lawrence W 160, 162 |
|   | Bibliographisches Institut 263 |
|   | Bic, Lubomir 107 |
|   | Bierwisch, Manfred 5, 32, 52, 192 |
|   | Bock, Kathryn 3, 45, 82, 88, 233 |
|   | Boddy, Mark 143 |
|   | Boyes-Braem, Penny 58 |
|   | Bradshaw, Jeffrey M 263 |
|   | Bratenberg, Valentino 148 |
|   | Brants, Thorsten 83, 85 |
|   | Buchholz, Kai 139, 140 |
| C | Carroll, Mary 42 |
|   | Carstensen, Kai-Uwe 26 |
|   | Chater, Nick 55 |
|   | Chen, Yihsiu 13, 237 |
|   | Chomsky, Noam 2 |
|   | Clark, Andy 145 |
|   | Crandall, Richard L 80 |
|   | Crocker, Matthew W 83, 85 |
| D | Dale, Robert 15, 17, 26, 33, 54, 55, 96, 194, 195, 205 |
|   | Davidson, Donald 8 |
|   | Davidson, Douglas J 45, 82, 88, 233 |
|   | de Ruiter, Jan Pater 13, 237 |
|   | De Smedt, Koenraad 15, 32, 55, 80, 83, 104, 156, 191, 192, 194, 195, 226, 235 |
|   | Deacon, Terrence W 62 |
|   | Dean, Thomas 143 |
|   | Dennett, Daniel C 146 |
|   | d’Inverno, Mark 236 |
|   | Dörner, Dietrich 61, 148 |
| E | Ebert, Christian 26 |
|   | Egg, Markus 109, 111, 112 |
|   | Eikmeyer, Hans-Jürgen 84, 85 |
|   | Endriss, Cornelia 26 |
|   | Erichsen, Mark 46 |
|   | Erk, Katrin 114 |
|   | Erwin, David E 45 |
|   | Eschenbach, Carola 19, 63, 65, 73, 110, 189, 198, 213 |
|   | Eysenck, Michael W 38, 158, 167 |
| F | Finkler, Wolfgang 80, 83, 86, 87, 144 |
|   | Fodor, Jerry A 28, 34, 35, 62, 150, 151, 269 |
|   | Ford, Kenneth M 275, 281 |
|   | Foth, Kilian 48, 105 |
| G | Gärdenfors, Peter 60–62, 65 |
|   | Gardent, Claire 112 |
|   | Gentner, Dedre 29, 236 |
|   | Gerhard, Christian 46 |
NAME INDEX

Markman, Arthur B 29, 236
Martin, James H 26
McCarthy, John 145, 146, 266
McClelland, James L 34
McDonald, David D 26, 33
McNeill, David 13, 274, 279
Medin, Douglas L 59, 60
Medweideff, Marion C 80
Menzel, Wolfgang 27, 48, 105, 145
Mercadal, Dennis 265
Mervis, Carolyn B 58
Meteer, Marie W 52, 194
Meyer, Antje S 29, 33
Meyer, David E 148
Miller, George A 167
Milnes, Brian G 117
Milward, David 55
Montague, Richard 63
Morgenstern, Leora 145, 146
Murphy, Gregory L 59, 179

– N –
Newell, Allen 18, 87, 146, 147, 158
Niehren, Joachim 109, 111, 112
Nijholt, Anton 277, 278
Norvig, Peter 18, 141
Nuyts, Jan 44, 274, 280, 283

– P –
Paul, Hermann 3
Pechmann, Thomas 54, 195
Pederson, Eric 274, 280, 283
Pianta, Emanuele 111
Pickering, Martin 55
Pinker, Steven 14, 53
Pop, Horia F 48, 105
Pullum, Geoffrey K 42
Pylyshyn, Zenon W 28, 275, 281

– R –
Ramalingam, G 88
Reiter, Ehud 15, 17, 26, 33, 54, 55, 96, 98, 112, 194, 195, 205
Reiter, Raymond 95, 145
Reithinger, Norbert 237
Reps, Thomas 88
Rickheit, Gerd 84, 85

Rieckmann, Bärbel 41
Robertson, David A 28
Roelofs, Ardi 29, 32, 33
Rosch, Eleanor 58
Rumelhart, David E 34
Russell, Stuart J 18, 141
Ryan, James L 80
Sandholm, Tuomas 141
Sapir, Edward 3
Schilder, Frank 110, 111, 115, 116, 225
Schopp, Andrea 32
Schreuder, Robert 5, 32, 52, 192
Schröder, Ingo 48, 105
Searle, John R 37
Shanahan, Murray 145
Shaw, Alan C 107
Simon, Herbert A 18, 141, 142, 146
Slobin, Dan I 3, 13, 25, 42, 47, 75, 242
Smith, Edward E 38, 59, 60, 179
Smith, Neil 2, 35
Spivey, J Mike 117–120, 122, 245, 247, 248
Strube, Gerhard 263
Taatgen, Niels A 147
Tannenbaum, Andrew S 107
Tappe, Heike 32, 45, 47, 48, 51, 52, 59, 165, 191, 276, 278, 284

– S –
Sapir, Edward 3
Schlicht, Gabriele 97, 251, 252
Schütz, Hubert 30
Schwarz, Jan 35, 148
Searle, John R 37
Shanahan, Murray 145
Shaw, Alan C 107
Simon, Herbert A 18, 141, 142, 146
Slobin, Dan I 3, 13, 25, 42, 47, 75, 242
Smith, Edward E 38, 59, 60, 179
Smith, Neil 2, 35
Spivey, J Mike 117–120, 122, 245, 247, 248
Strube, Gerhard 263
Taatgen, Niels A 147
Tannenbaum, Andrew S 107
Tappe, Heike 32, 45, 47, 48, 51, 52, 59, 165, 191, 276, 278, 284

– V –
van Benthem, Johan 68
van Deemter, Kees 195
van der Sluis, Ielka 195, 237
van Erk, Sebastiaan 195

289
NAME INDEX

Verleg, André 195
Vilain, Mark B 68, 69
von Humboldt, Wilhelm 2
von Kleist, Heinrich 2
von Stutterheim, Christiane 41, 277, 284

– w –
Wahlster, Wolfgang 112
Weiß, Petra 84, 85
Whorf, Benjamin Lee 3, 42
Wiese, Heike 31, 42, 192
Wilson, Robert A 57, 277, 280, 283, 284
Wirén, Mats 83
Wordsworth, John B 117
Wundt, Wilhelm 2, 3, 82

– z –
Ziesche, Soenke 19, 32, 55
Zilberstein, Shlomo 143, 264
Zock, Michael 15, 191
SUBJECT INDEX

priming, 53, 54, 44, 44, 162, 172, 238
principle of natural order, [5]
process, 8, 66, 80, 82, 82, 83, 96, 98, 100
as means of structuring models, 31, 56, 87, 88 as situation type, [8]
processing, automatic, 36, 45, 55, controlled, 35, 37, parallel, 36, piecemeal, 45, 83
sequential, 46, volitional, 85
proposer, 197
production, 47, 49, 76, memory, [47]
Prolog, 129
property, 41, 52, 65, natural, 62
propositionalisation, 67, 62
prosody, 53
P.S., 148
psycholinguistics, 35
PVM-generation, 35, 9, 16, 17, 31, 41, 47
57, 63, 55, 93, 97, 98, 152, 159, 167
159, 160, 161, 162, 181, 180, 190
191, 196, 192, 215, 218, 220, 223
228, 229, 253, 256, 258
Q
quasi-module, 31, 52, 66, 82, 155, 158, 164
239
R
racing, 59, 127, 235
rationality, bounded, 55, 88, 95, 142, 144
unbounded, 141, 142
reactivation, 110, 160, 236
real-time, 208, 223
reasoning, 35
recognise–decide–act cycle, 147, 149
referential nets, 0, 9, 20, 28, 28, 52
53, 60, 83, 92, 93, 103, 112, 119, 118
123, 128, 129, 132, 133, 135, 141, 419
209, 205, 208, 209, 212, 230, 231
referential object, see refO
referring expression, 14, 25, 25, 46, 165
189, 207, 208
refO, 19, 20, 25, 35, 66, 68, 72, 73, 82, 88
91, 93, 97, 100, 111, 115, 122, 128, 131
133, 135, 152, 157, 160, 165, 169
171, 183, 185, 188, 191, 195, 200
203, 205, 209, 211, 212, 215, 216, 218
221, 223, 225, 226, 228, 238, 241, 258
discourse, 204, temporary, 129
verbalisation, 239, 130, 164, 165
167, 169, 176, 189, 200, 203, 210, 213, 218
209, 215, 258
region, 62, 212
representation, 14, 34, 63, 64, 68, 83
conceptual, 0, 9, 11, 16, 28, 31
111, 12, 48, 59, 62, 51, 50, 57, 79, 84
88, 93, 116, 116, 116, 128, 158
128, 154, 175, 202, 203, 205, 233, 234
337, correctness of, 167, 68, dynamic, 31, 147, 19, 19, 241, embodied, 67
337, event, 167, 19, hierarchical, 60, 10
17, 17, 20, 20, 20, 20, 240, imagistic, 38
internal, 17, 20, 20, layer, 59
objective, 67, pre-linguistic, 32, 42
propositional, 31, 27, 27, 28, 55, 56
semantic, 3, 14, 15, 20, 20, 112, 113, 114
193, subjective, 19, visuo-spatial, 29
resource, 11, 63, 33, 38, 54, 55, 66, 67
83, 84, 93, 98, 104, 106, 117, 116, 137
151, 146, 151, 220, 234, 240, 241, 243
240, run-time, 54, 72
81, 89, 92, 102, 103, 143, 149, 157, 194
229, 223
robustness, 101, 103, 103, 108, 113, 145, 146
156
salience, 62, 119, weight, 62
satisficing, 1, 42, 241
scheduling, 106, 107
segmentation, 107, 107
selection, 5, 8, 9, 11, 13, 14, 47, 48, 50, 52
53, 54, 60, 88, 89, 91, 93, 97, 98, 150
153, 156, 154, 150, 164, 167, 175
178, 188, 189, 190, 195, 196, 211, 212
215, 214, 233, 236, 238, strategy, 49
50, 81, 142, 189, 190, 190, 208, 211
212, 215, 216, 218, 219, 235, 236, 238
sequentiality, 106, 107
serialisation, 49
sign language, 16
similarity, 60, 61, 72, 81
situation, 1, 42, 160, 165, 189, 190, 198
199, 204, 215, 213, 215, 216, 218, 219
224, 258, ongoing, 73
situation calculus, 1, 45
sketch maps, 29, 41, 46, 48, 160, 165, 175
296
Ich erkläre hiermit an Eides statt, daß ich diese Arbeit selbst verfaßt und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Hamburg, im Oktober 2003

Markus Guhe