

**Incoherence and Text Comprehension:
Cognitive and Computational Models of Inferential
Control**

by

Elliot Smith

A thesis submitted to the Faculty of Science
of the University of Birmingham
for the Degree of
Doctor of Philosophy

School of Computer Science
Faculty of Science
University of Birmingham
September 2000

UNIVERSITY OF
BIRMINGHAM

University of Birmingham Research Archive

e-theses repository

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

Abstract

This thesis describes work on defining and modelling text comprehension. The basis of my approach is a theory of comprehension as a form of abductive reasoning. The specific problem addressed is inferential control and the management of alternative, competing representations.

This problem is related to issues of representation quality, with decisions between representations being made on the basis of quality comparisons. Simultaneously, monitoring of representation quality determines when comprehension halts; in other words, there is some kind of threshold against which quality is compared.

In the first part of the thesis I analyse concepts of representation quality, describing the structure of episodic and semantic representations and processes. I then look at metrics for representation quality before developing my own metric. The metric is based on the concept of incoherence, derived from the structural potential of representations.

The second part of the thesis describes a computational model of incoherence, the Incoherence-Driven Comprehender (IDC). IDC combines AI implementation technology with insights from cognitive psychological studies of text comprehension. I show how IDC can be applied to various comprehension tasks.

Throughout the thesis I suggest how aspects of IDC's architecture and behaviour may offer a fresh perspective on human comprehension.

Keywords: abduction, coherence, episodic and semantic memory, inference, interpretation, representation quality metrics, story understanding.

Acknowledgements

Thanks and love to Nicola for her patience and support. I couldn't have done it without her.

Thanks also to my supervisor, Peter Hancox, for his guidance and perspective; and to my thesis group members, Mark Torrance and Ela Claridge, for their enthusiasm and helpful comments.

Thanks to my family and friends for their encouragement.

Special thanks to Jeremy Sproston, Kevin Lucas, Gavin Brown and Adrian Hartley for technical assistance.

Contents

1	Introduction	1
1.1	Approaching Comprehension	1
1.2	A Broad Overview of My Work	3
1.3	Incoherence-Driven Comprehension	4
1.4	Research Methodology	6
1.5	Structure of the Thesis	8
2	Background to the Problem	9
2.1	A Brief History of Text Comprehension	9
2.2	Implementing Constructive Comprehension	10
2.2.1	A Note on Notation	12
2.3	Inferential Promiscuity	13
2.3.1	Bi-directional Chaining	13
2.3.2	Incomplete Matching	17
2.3.3	Number of Rules	18
2.3.4	Can Inferential Promiscuity be Prevented?	19
2.4	Inferential Control	20
2.4.1	Chapter Summary	27
3	Episodic Representations and Inference	28
3.1	What’s the Point of Comprehension?	28
3.2	Computing Comprehension	31
3.2.1	Formalising and Visualising Comprehension	33
3.2.2	Are Interpretations Just Explanations?	36
3.2.3	Can Text Propositions Explain Each Other?	43
3.2.4	Should All Observations be ‘Explained’?	44
3.2.5	Are Multiple Representations Maintained?	46

3.2.6	Chapter Summary	58
4	Semantic Representations	59
4.1	The Semantic/Episodic Distinction	59
4.1.1	Overview of Schemas	61
4.1.2	Schemas and Logic	63
4.2	Scripts	65
4.2.1	Are Scripts Too Top-Down?	66
4.2.2	Scripts and Information Accessibility	68
4.3	Associative Networks	70
4.3.1	Control and Marker Passing	72
4.3.2	Are Schemas Necessary?	76
4.3.3	Benefits of Schemas	77
4.4	Chapter Summary	80
5	Metrics for Comprehension	81
5.1	Representation Quality	83
5.1.1	Specificity	83
5.1.2	Simplicity	85
5.1.3	Breadth	88
5.1.4	Competitiveness	89
5.1.5	Probability	91
5.2	Coherence	100
5.2.1	Coherence in the Text	100
5.2.2	Coherence in the Representation	101
5.2.3	Local and Global Coherence	102
5.2.4	Quantifying Coherence in the Representation	103
5.3	Incoherence	107
5.3.1	Basic Concepts	107
5.3.2	Incoherence in IDC	110
5.3.3	Computing Informativity and Ubiquity	114
5.3.4	Incoherence of Instances	116
5.3.5	Incoherence of Trees	118
5.3.6	Skepticism	122
5.3.7	Incoherence Change	128

5.3.8	Specificity and Incoherence	129
5.3.9	Simplicity, Breadth and Incoherence	131
5.3.10	Competitiveness and Incoherence	132
5.4	Chapter Summary	132
6	The Incoherence-Driven Comprehender	135
6.1	Semantic Representations: Schemas	136
6.1.1	Eventuality Types	136
6.1.2	Roles	137
6.1.3	Constraints	138
6.2	Semantic Representations: Texts	140
6.3	Episodic Representations	141
6.3.1	The Short-Term Store (STS)	141
6.3.2	The Long-Term Store (LTS)	143
6.3.3	Interpretations and Representations	143
6.4	Comprehension Processes	145
6.4.1	Creating New Extensions	146
6.4.2	Updating the Interpretation	151
6.4.3	Psychological Correlates of Range	152
6.5	Chapter Summary	154
7	Examples of IDC's Behaviour	156
7.1	Plan Recognition	157
7.1.1	An Illustrative Example	159
7.1.2	General Discussion	166
7.2	Inference Protocols	167
7.2.1	Making Sense of the Protocol	171
7.2.2	Time Course and Content of Comprehension	174
7.2.3	General Discussion	178
7.3	Accounting for 'Role-Shift' Texts	183
7.3.1	Revision of Representations	184
7.3.2	Multiple Representations	187
7.3.3	Implications for Human Comprehension?	190
7.3.4	General Discussion	197
7.4	Chapter Summary	202

8	Conclusion	204
8.1	Evaluation and Summary of Research	204
8.2	Evaluation of IDC	206
8.2.1	Limitations of Relations	206
8.2.2	Lack of Consistency Checking	207
8.2.3	Problems with Comprehension Management	209
8.2.4	Problems with the Metric	210
8.3	Extensions	212
8.3.1	Increasing the Formality of the Metric	212
8.3.2	Insights from Analysis of Executive Function	213
8.4	Achievements	214
8.5	Last Words	215
A	Example of Tree Creation	217
B	IDC Schema Code, Texts and Examples	222
B.1	Plan Recognition	222
B.1.1	Plan Recognition Schemas	222
B.1.2	Example Plan Recognition Trees	229
B.2	Inference Protocols	232
B.2.1	Ivan Story Text	232
B.2.2	Ivan Story Schemas	233
B.3	Role Shift Texts	242
B.3.1	Janitor Story Text	242
B.3.2	Janitor Story Schemas	243
C	IDC Program Code	248
C.1	Running IDC	248
C.1.1	Preparing IDC's Knowledge Base	249
C.1.2	Running the Comprehension Simulation	249
C.2	<i>IDC_setup.pl</i>	251
C.3	<i>IDC.pl</i>	263
C.4	<i>shared.pl</i>	314
	Bibliography	333

List of Figures

2.1	The necessity of bi-directional inference	16
2.2	Generic process model for the inference cycle	21
3.1	Episodic network representation of an abductive explanation	35
3.2	A tree representing a script-like explanation	37
3.3	An elaborative tree	38
3.4	Causal antecedent vs. causal consequent inferences	40
3.5	Representation of the ‘fly swatting’ text	41
3.6	Inferences connecting newspapers and fly swatting	42
3.7	Possible structures for an interpretation	48
3.8	A labelling of an ATMS	49
3.9	A representation of the meaning of ‘river bank’ (after [Kintsch, 1988]) . . .	55
3.10	Activation profile for concepts involved in comprehending ‘river bank’ . . .	56
3.11	Activation profile for concepts involved in comprehending ‘the bank collapsed’	57
4.1	Representing rules as a semantic network	69
4.2	Alternative semantic representations of the same knowledge	71
4.3	Pronoun resolution network (figure 7 of [Kintsch, 1988])	79
5.1	Alternative explanations for ‘gaining weight’	87
5.2	(a) Explanations without competition; (b) Explanations with competition (after [Read and Marcus-Newhall, 1993])	90
5.3	A Bayesian network representation of a text	94
5.4	Representation of equality between restaurants	97
5.5	How ACCEL fails to account for potential implications	105
5.6	The role of the knowledge base in measuring coherence	106
5.7	Actualising potential representations reduces the PRS	113
5.8	A simple schema lattice with labels	115

5.9	Calculation of <i>instance_inc</i>	119
5.10	Calculation of <i>altelabs_inc</i>	120
5.11	Calculation of <i>altexpls_inc</i>	121
5.12	An incoherence decision point: Skepticism governs whether an explanation is inferred	126
5.13	Three representations for ‘Sharon went to the supermarket’	130
7.1	Adapted Ivan Story protocol diagram (after [Trabasso and Magliano, 1996])	170
7.2	Multiple representations of the Janitor Story (Skepticism = 0.5, Range = 1)	189
7.3	Relative time spent on sentences of the non-role-shift-text (text 1), Skep- ticism = 0.5	195
7.4	Relative time spent on sentences of the role-shift-text (text 2), Skepticism = 0.5	195
7.5	Relative time spent on sentences of the non-role-shift-text (text 1), Skep- ticism = 0.1	196
7.6	Relative time spent on sentences of the role-shift-text (text 2), Skepticism = 0.1	196

Chapter 1

Introduction

1.1 Approaching Comprehension

This is what things have come to in this world
The cows sit on the telegraph poles and play chess
(from ‘End of the World’ (1916) by Richard Huelsenbeck, in [Richter, 1965])

Human conceptual activity has a ‘requirement for understanding’ at its core. As our senses continually gather data, we are driven to catalogue, compress and store it. When confronted by a novel group of stimuli, we try to explain how those stimuli arrived at our senses, where and how they originated, why they happened to be grouped together, and so on. This process of constructing interpretations can be described by the general term *comprehension*. This term applies both to tasks colloquially described as comprehension, such as reading, and to other tasks often considered as separate activities, such as categorisation and visual perception. In all of these cases, the comprehension process generates an *interpretation*, a representation which depicts, describes, models or ‘stands for’ the data which was observed.

Text comprehension is a particularly specialised aspect of this general cognitive ability. While it has similarities with other forms of comprehension, there are peculiar complications. Most of these are derived from a statement which is (generally) true of texts, but not of eventuality sequences in the real world: a text is (generally) *composed* with the purpose of *conveying meaning*.¹

¹I use the term *eventuality* to refer to events and states, after [Moens and Steedman, 1988].

This underlying purpose often leads to the mistaken assumption that the comprehender's aim is to uncover a text's 'intended meaning'. However, the only *warranted* assumption is that everything mentioned in a text contributes to its meaning; the elements of a text are relevant with respect to each other. In turn, this implies that relationships between elements of a text are intended (though the exact nature of those relationships is not predetermined); and that finding relationships between these elements is desirable. Thus, an interpretation which connects elements of the text is preferable to an interpretation which leaves them disparate. In other words, the more *coherent* an interpretation is, the better.

The above summary of comprehension is, naturally, incomplete and not uncontroversial. It glosses over several difficult questions:

1. How do comprehenders make connections between elements of the text?
2. How does a comprehender know when to stop making connections?
3. Are the connections formed by the same mechanisms used for generating connections between real world event sequences?

These issues have been approached by previous researchers, as I describe in section 2.1. However, during the history of this research, problems which were not initially obvious came to light, mainly as a result of implementation efforts (see section 2.2). Because implemented models had to be very detailed to run as programs, every aspect of the comprehension process was minutely analysed. An obvious first step was to program computer models to make inferences from a text: no text explicitly contains the entire world in which it takes place, and every instance of comprehension involves inference. However, when you allow a program to make inferences, the problem of *inferential promiscuity* soon becomes apparent: the tendency for unrestrained processing to give rise to spurious inferences which tax the credibility and/or usefulness of an interpretation (see section 2.3).

Thus, a processing constraint on comprehension was recognised: some limit on inference making had to be imposed to prevent systems from wasting energy and time on promiscuous inferences. Presumably, in a psychological (human) context, a similar control mechanism must be at work, otherwise the comprehender would be inundated with the implications of a text and unable to form viable interpretations. Therefore, some relevancy or utility criteria must be at work. In a computational context, these criteria were naturally implemented using heuristic methods for restricting inferences and evaluating interpretations.

The issue of constraints on inference-making provided the initial impetus for my research and remains central to this thesis. My approach to this issue has been the development and implementation of a comprehension model which incorporates several cognitively-motivated control mechanisms. The next two sections briefly describe my work, to give an idea of important concepts. I then describe my general research methodology in section 1.4. Finally, section 1.5 explains the structure of the thesis, hopefully showing the ‘thread’ of the argument as a whole.

1.2 A Broad Overview of My Work

As I mentioned in the previous section, the problem of how to constrain inferences during comprehension provided the initial impetus for my work. In particular, I was interested in how a comprehender’s subjective assessment of their current interpretation could influence future inference processes. Two particular aspects of the problem became evident:

1. *Inferential Decisions*

Decisions about which inferences to make depend on some method of comparison. Given two possible inferences, how can a comprehender decide which to make and which to discard? Or whether both should be made in parallel?

2. *Comprehension Halting*

During comprehension, why/when/how does the comprehender decide to stop enriching their current interpretation and ‘accept’ it? Presumably, any amount of information could be added to an interpretation; so why don’t comprehenders produce exhaustive interpretations?

My approach was to adapt an idea from the field of abduction metrics in AI research, with the aim of modelling inferential processing in human comprehenders. Abduction is a method of inference often used in modelling explanation generation in AI (e.g. [Ng and Mooney, 1990], [Peng and Reggia, 1990]); more recently, cognitive psychologists have suggested abduction as an important mechanism in human comprehension [Noordman and Vonk, 1998]. In computational implementations, the nature of abduction often leads to competition between explanations (see section 3.2.5). This competition has been solved by rating explanations according to metrics which refer to appealing concepts like simplicity, probability and coherence (see chapter 5). The explanation selected from

the set of competitors is the one which is assigned the highest ‘quality’ rating by the metric (e.g. it is the simplest, most probable, or most coherent explanation).

In my work, such a metric is used for the rating of representations, based on a notion of *incoherence*. Representations are rated according to the amount of incoherence they contain; this is assigned by determining the total *potential structure* of the representation’s elements. Inferences can also be rated, by measuring how well they increase the coherence of a representation by reducing its incoherence (c.f. [van den Broek et al., 1995]).

While incoherence of elements was fixed with respect to a particular knowledge base, I decided to modulate incoherence by subjective, process-based criteria. My first step was to incorporate a *skepticism* parameter: by altering the model’s skepticism, the same text can be read with the same knowledge base on different occasions, yet produce different comprehension profiles. For example, the system may make inferences at different points in time, or even make different inferences. Some examples are given in chapter 7.

Later refinements to the model added further control mechanisms, to allow it to cope with short- and long-term memory stores and maintenance of multiple representations (see chapter 6).

The next section defines the main characteristics of the theory.

1.3 Incoherence-Driven Comprehension

The model and implementation described in this thesis go under the collective heading of IDC (Incoherence-Driven Comprehender). IDC is intended as a generic theory of text comprehension which builds on much past work: I do not introduce too many complicated new representational formalisms or algorithms, but use several existing formalisms as a foundation. The theory is also, of necessity, simplified and abstracted away from low-level morpho-syntactic processes, focused instead on semantic and pragmatic issues. I have also side-stepped several important high-level elements of comprehension, such as the ability to learn and the role of creativity. However, I recognise the importance of all of these missing factors. In an ideal model of comprehension, parsing, learning and creative inference would all play important roles; unfortunately, such an ideal is not realistic within the bounds of a PhD.

The particular focus of the work is *inference making*. As I describe in chapter 2, inference making has been a central concern of comprehension modelling for many decades, being the dominant process in the construction of interpretations. Within the topic of

inference making, the role of control in comprehension has been central to my work. Control is of concern both in cognitive and computational systems, yet is often glossed over in discussions of comprehension. My aim is to address this imbalance, suggesting possible interlaced control mechanisms which act together in the management of inferences.

Control in IDC has its basis in a theory of *incoherence*. A first definition of incoherence will be useful at this point (and will be progressively refined throughout the thesis). I intend the word ‘incoherence’ in a sense which diverges from its intuitive definition. In everyday conversation, incoherence is defined as ‘a lack of clarity’ or ‘lack of organisation’ [Collins English Dictionary, 1994]. In IDC, the definition of incoherence is less intuitive, yet hopefully more precise:

The incoherence of a representation is determined by the number of representations which could possibly be derived from it via inference, or its *possible representations space* (PRS). The PRS of a representation depends on the comprehender’s knowledge base: if knowledge sources exist which could be applied to elements of the representation to produce new elements, new representations based on those elements are possible; if such knowledge sources do not exist, there are no other possible representations.

Note that this definition departs from the colloquial meaning of ‘incoherence’ as it measures incoherence with respect to a knowledge base, rather than with respect to individual texts. Human comprehenders tend to judge as incoherent those *texts* for which they are unable to form sensible interpretations. However, in IDC, the emphasis is on the role of *knowledge* in assessing incoherence: it is the lack of a suitable knowledge structure which prevents construction of a sensible interpretation, rather than the text. If a knowledge structure could be brought to bear on the text and thus allow it to be integrated, the incoherence would disappear.

Even though this definition seems to deny that *texts themselves* are coherent or incoherent, the argument can be adjusted to show that this is not the case. Judgements of the coherence and incoherence of particular texts are made with respect to a shared reserve of knowledge structures, a kind of cultural knowledge base (e.g. English sentence structure, Western genre conventions); those texts which can be integrated and structured by that knowledge base are coherent, while those which cannot are incoherent. For example, Agatha Christie’s novels seem neat and coherent to Western readers, as the plots, characters and language merge snugly with recognised, stereotyped knowledge structures

common to those readers. However, it is easy to demonstrate the relativity of judgements of coherence by examining texts from other cultures, or texts from subcultures within a culture such as the example at the start of this chapter.² These texts only ‘make sense’ or can be made coherent with respect to certain knowledge structures: an understanding of shared imagery, the significance of particular plot devices, multiple meanings of particular words, outlooks on the world, etc..

Comprehension and the assessment of coherence are thus chiefly dependent on two aspects of the *comprehender’s context*:

1. The amount and types of knowledge (in an abstract sense) which the comprehender brings to the text. This includes their specific, personal knowledge and shared, cultural knowledge.
2. The abstract evaluation of representations, based on the comprehender’s subjective criteria for an acceptable interpretation (their *skepticism* and *tolerance of incoherence*).

The theoretical background to this theory is described in greater depth in section 3.1. The theory of incoherence is described in chapter 5, while the full implementation of IDC is described in chapter 6.

In the next section, I discuss the methodological framework within which this work was carried out.

1.4 Research Methodology

Psychological models are often concerned with describing the relationship between psychological data and a generalised, natural language description. The need to cope with real-world data means that implementations frequently have to be curtailed and drastically simplified: for example, the stimuli involved may be reduced to propositional representations (rather than first-order predicate calculus formulae). While these simplifications are necessary, they can mean that it is sometimes difficult to see how the theory would be extended to more complex data.

²Bartlett’s early experiments on memory demonstrated the tendency for comprehenders to lever an unfamiliar text (a folktale called *The War of the Ghosts*) from a foreign culture (Indian) into culturally-familiar structures [Gardner, 1987].

Frequently, this drive to simplification may also result in dynamic processes (such as construction of an interpretation) being split into their component parts. For example, these might include ‘deciding whether to extend an interpretation’ [van den Broek et al., 1995], ‘deciding which elements to retain in working memory’ [Fletcher et al., 1990], and ‘rating interpretations’ [Thagard, 1988]. While it is often necessary to split processes in this way, it can obscure the relationships between them and remove the need to define the interactions between different parts of a process. For example, decisions about whether to extend an interpretation may depend on the goodness of that interpretation; in turn, this depends on the rating mechanism used; and again, deciding which elements to retain in working memory may depend on how useful they may be for understanding future elements.

On the other hand, AI models are usually detailed and integrated: processes and the interfaces between them are specified in great detail, as the program actually has to ‘run’. Each process must function correctly and produce an output usable by related processes; every piece of information involved in the task has to be defined; there is no way to gloss over processes if they are inconveniently complex. However, this drive to specification can give rise to an underlying yearning for ‘correctness’ of the model. This means that often a lot of emphasis is given to logical consistency and computational efficiency, such that psychological evidence can go out of the window (e.g. Ng and Mooney [Ng and Mooney, 1990]). By concentrating on ‘proof’, AI models can also become highly restricted: the range of phenomena they deal with is reduced to increase the speed or correctness of the algorithm.

In my model, I have attempted to find a middle-ground between these two approaches, possibly leaning more towards AI. So, while I attempt to define the mechanisms that might underly the choice between two explanations in a psychological experiment, I have done so with a computational theory (abduction). And while I have aimed to make the theory as ‘correct’ as possible (in a computer science sense), it is impractical as a ‘commercial’ solution. For example, the data structures used in the rulebase are limited in their expressive power, as they were deliberately designed to be as ‘process-free’ as possible. This means that some efficiency is lost, but hopefully with a concurrent gain in clarity.

I have also tried to produce a theory which is wide-ranging and takes into account much current cognitive psychological research, while using the minimum number of new theoretical constructs. Those constructs I do introduce are set against a background of sev-

eral well-established ideas, such as scripts [Schank and Abelson, 1977], working memory ([Shallice, 1982], [Fletcher et al., 1990]), and coherence-seeking, constructive comprehension ([van den Broek et al., 1995], [Graesser et al., 1994]).

1.5 Structure of the Thesis

- Chapter 2 covers the general background to my work. It puts in place the historical context, and a general theoretical framework for analysing story comprehension theories.
- Chapters 3 and 4 specify a framework for modelling comprehension, both in cognitive and computational terms. This framework is then used as a rationale for my own computational model.
- Chapter 5 concerns the central problem defined in this introduction: how a comprehender can judge the relative quality of competing representations, and how representations are managed within the context of interpretation. My answers are chiefly computational, as is reflected by the tone of the chapter.
- Chapter 6 describes my implemented model. The intention is to show how the model maps onto the framework described in the previous chapters; I also suggest some cognitive analogues of mechanisms in the model.
- Chapter 7 contains some tentative experimental results, with some suggestions for comparing the output of the model with human behaviour.
- Chapter 8 discusses the achievements of my work, as well as places where it fails. I also suggest some directions for future work.
- The three appendices cover actual program code and examples of output. These have been removed from the main body of the thesis to improve its readability.

Chapter 2

Background to the Problem

This chapter describes the broad theoretical background to text comprehension as a whole. It then covers in greater detail the particular question which prompted this thesis: how can inferences during comprehension be controlled?

2.1 A Brief History of Text Comprehension

Text comprehension has already received much attention in many academic fields over a long period of time. For example, some literary critics, such as Roman Ingarden and Wolfgang Iser, have examined the interpretory processes involved in the act of reading [Ingarden, 1973], [Iser, 1971]; and several psychologists have tried to specify and demonstrate the existence of the knowledge structures underlying story recall [Bartlett, 1932]. However, these analyses typically take the form of informal verbal descriptions of the mechanisms involved. This informality is not a deficiency: it is simply an artifact of the times in which they were written and contemporary research paradigms.

Psychologists were the first to concern themselves with the *detailed* processes comprising text comprehension. During the reign of Behaviourism in the 1920s, 30s and 40s, many psychologists avoided discussing mental representation: psychological theories were to be based on observable evidence alone, without recourse to dubious hypothetical mental entities [Gardner, 1987]. However, by the early 1950s, many researchers realised that Behaviourism was an intellectual dead-end, unable to provide reasonable or realistic theories of language, planning, or other aspects of the internal organisation of external behaviour. As a result, these psychologists turned to descriptions of comprehension which explicitly defined individual mental components. These researchers found useful metaphors for

their work in the emerging fields of computer science and cybernetics: for example, Miller described human short-term memory capacity in terms of information theory, using *bits* as a measure of capacity [Miller, 1956]; and Broadbent regarded whole organisms as information-flow mechanisms, depicting them as flowcharts resembling program designs [Broadbent, 1958].

This shift to *cognitive* psychology, influenced by advances in computer science, led to the use of computer programs as models of human behaviour. Some of the earliest programs to deal with language comprehension issues include Bobrow’s STUDENT system for solving algebra problem stories [Bobrow, 1968] and Quillian’s system for knowledge retrieval from semantic networks [Quillian, 1968]. Both systems had to deal with the problem of *inference*: deriving implicit information from explicit data. For example, STUDENT could infer the antecedents of pronouns and Quillian’s system could infer conceptual overlap between discrete input units.

It was obvious that inference was central to comprehension: construction of an interpretation requires relating inputs to stored knowledge, such that any ‘gaps’ in the input are filled and connections between input units specified. No text contains all the information required for its understanding, but gives ‘cues’ which facilitate construction of its ‘meaning’; comprehension involves a ‘search after meaning’ where inferences build representational units in memory, relating and grouping input units [Graesser et al., 1994]. This *constructive* approach to comprehension has its basis in Bartlett’s theory of *schemas* [Bartlett, 1932], and was highly influential in the 1970s, particularly with cognitive scientists, e.g. [Charniak, 1972] and [Rumelhart, 1975]. These 1970s researchers were trying to find models of human text comprehension precise enough to be implemented as computer programs; in the next section, I describe their computational work as it forms the background and impetus for my own research.

2.2 Implementing Constructive Comprehension

The models of text comprehension developed in the 1970s shared a concern with the constructive role of inferences. An important issue was how to model the origins of these inferences. The answer lay in adapting Bartlett’s schemas to represent stored knowledge.

Schema theory states that when new data arrives at the senses, it activates stored schemas which are used to encode that data in memory. For example, on seeing an agglomeration of lines and colours in some particular alignment, the schema for ‘bird’ may

be activated. Once a schema has been activated in response to an input, the *slots* attached to the schema become available for ‘filling’. Slots are blank ‘conceptual spaces’ into which actual instances are inserted as they are encountered in the input. Alternatively, slots may be filled by default inferences: for example, the slot for the type of instrument used in a *coffee_stirring* schema may be filled by the default value *spoon*. More details about models based on schema theory are given in chapter 3.

The computational realisations of Bartlett’s ideas resulted in various schema-like structures variously called ‘frames’ [Minsky, 1975], ‘scripts’ [Schank, 1975], ‘macrostructures’ [Correia, 1980], and even ‘schemas’ (a direct borrowing from Bartlett) [Rumelhart, 1975]. Distinctions were frequently made between these structures, but at heart they are very similar (Minsky himself compared his conception of frames to scripts). For this reason, I will use the generic term *schemas* to refer collectively to these structures. For the moment, I discuss them generally, leaving detailed discussion of their technical characteristics until section 4.1.1.

Early AI systems based on schema theory were generally implemented as production systems. In these systems, schemas were written as rules which represented implications from a higher-level structure to one or more lower-level structures. This represents the assumption that superstructures are abductively implied by the presence of their substructures; for example, observing someone entering a bank allows a comprehender to abductively infer that they plan to deposit some money.¹ New elements are added to the representation by matching current elements to the right-hand side (or *consequent*) of a rule; the left-hand side (the *antecedent*) is then *inferred* by backward-chaining and asserted as a new element. An example rule in this format is:

$$(1) \textit{hold_plan}(\textit{planner}:X, \textit{object}:Y) \longrightarrow \\ \textit{grasp}(\textit{agent}:X, \textit{object}:Y).$$

This rule specifies that if some actor X grasps some object Y , a reasonable inference is that X has a plan to hold Y . Another rule in the same format is:

$$(2) \textit{goal}(\textit{planner}:X, \textit{objective}:(\textit{possess}(\textit{agent}:X, \textit{object}:Y))) \longrightarrow \\ \textit{hold_plan}(\textit{planner}:X, \textit{object}:Y).$$

¹c.f. inferring causes from effects in medical diagnosis, as in PATHEX\LIVER in [Josephson and Josephson, 1996].

This rule specifies that if some planner X has a plan to hold Y , a reasonable inference is that X has a goal to possess Y . (Both rules are adapted from chapter 8 of [Schank and Riesbeck, 1981].) If a system with these two rules were presented with an input like:

grasp(agent:geoff, object:thing1)

two inference steps are possible (with the second being derived from the first):

1. *geoff* is planning to hold *thing1*.
2. *geoff* has a goal to possess *thing1*.

The above inferences are reasonable, given the input. The early systems formed chains of such inference steps between input elements, following the maxim ‘To “understand” is to establish relations between the new and the old’ [Schank, 1975]. This enabled them to perform well on simple texts.

However, part of the reason for this success was that the knowledge base consisted of very few rules and was geared towards a small group of similar texts. This meant that very little information could be derived, even if all the rules were applied; it also made it likely that derived information would be (at worst) marginally relevant, as the rules were designed for comprehending a collection of similar texts.

In summary, the small scale of the programs was partly responsible for their success. As production systems in other areas of AI grew, researchers began to realise that as the scale of the knowledge base increases, the number of potential inferences soon gets out of hand [Davis, 1980], [Georgeff, 1982], [Clancey, 1983]. Other problems arise due to the types of inference allowed, and the stringency (or lack of) with which inferences are made. These issues are described in section 2.3.

2.2.1 A Note on Notation

Note that the format for representing ‘knowledge structures’ (e.g. rules, scripts, schemas) is as consistent as possible throughout this thesis, and the rules in the previous section are no exception. Knowledge structures are assumed to have the form:

$$a \longrightarrow b_1, \dots, b_n.$$

This rule states that there is some relationship between an element a and some other elements b_1, \dots, b_n ; the commas in the consequent are intended as denoting something akin to the logical ‘and’ (\wedge). Elements may be either propositions, or may contain variables; variables are treated as they are in Prolog, and denoted by capital letters. (N.B. no quantifiers are used in rules.) Another assumption I have made is that events are composed of a predicate and a set of roles (c.f. [Alterman and Bookman, 1990], [Fillmore, 1968], [Lindsay and Norman, 1977]).

The exact nature of rule types and components is left for discussion in later chapters.

2.3 Inferential Promiscuity

Borrowing a term from Norvig and Charniak, an excessive number of inferences could be termed *promiscuous*, as it wastes the comprehender’s resources: time and memory ([Norvig, 1989], [Charniak, 1986]). These resources are wasted because chains of inference may be pursued exhaustively to irrelevant levels of detail and spuriousness. This requires commitment of resources to producing, storing, comparing, and removing representational elements, many of which may ultimately prove to be useless.

In production systems, the promiscuous inference problem is exacerbated by three main factors:

1. Bi-directional chaining.
2. Incomplete matching.
3. Number of rules.

Each of these can justifiably be added to a production system to increase its power; however, each can also increase the number of possible inferences. Each factor is described in more detail in the following sections.

2.3.1 Bi-directional Chaining

So far, the system I have described makes inferences by *backward-chaining* on rules: the consequent of a rule is matched against the input and its antecedent inferred [Frost, 1986]. An alternative is to *forward-chain* on rules. To do this, the system matches the antecedent

of a rule against the input and infers its consequent. This is the classical, familiar form of logical inference represented by the syllogism:

$$\frac{a, a \rightarrow b}{b}$$

If we again read the rules causally, forward-chaining allows inferences from causes to their effects; or, in other words, *prediction* [Shanahan, 1989]. In the system I'm outlining here, this might allow the following pattern of inference:

Observation:

goal(planner:emma, objective:(possess(agent:emma, object:orange1))).

Rule:

goal(planner:X, objective: (possess(agent:X, object:Y)) \longrightarrow
hold_plan(planner:X, object:Y).

Inference:

hold_plan(planner:emma, object:orange1).

Is the inference ‘*emma* plans to hold *orange1*’ useful? It doesn’t explain why *emma* has a goal to possess *orange1*; it merely predicts how she may go about fulfilling this goal. In this respect the inference *elaborates* the observation and doesn’t add to the understanding of ‘why’ *emma* acts the way she does. This seems to contradict recent psychological results, which contrast the redundancy and inefficiency of predictive inferences with the over-riding importance of explanatory (abductive) inferences [Graesser et al., 1994], [Keefe and McDaniel, 1993], [Trabasso and Magliano, 1996].

However, in the system we are considering, we can justify forward (–chaining) inference as necessary for three reasons, described below.

Coherence dependent on forward inference

While some researchers view forward inferences as being only minimally made, or as not being drawn at all (see [Keefe and McDaniel, 1993] for a review), others have shown that forward inferences do play a part in comprehension. Some research shows that forward inferences are made, but that they are quickly removed from working memory unless immediately confirmed by subsequent text [Keefe and McDaniel, 1993]. In some cases, they may even establish ‘predictive explanations’ for statements currently in working memory [Murray et al., 1993]. Seemingly inexplicable eventualities may only be

explainable by reference to some expected future eventualities, for example (adapted from [Murray et al., 1993]):

The angry waitress was totally fed up with her job. When a rude customer criticised her, she lifted a plate of spaghetti above his head.

These two sentences do not seem to make sense (at this point in comprehension) without the causal consequent inference ‘She poured the spaghetti over the customer’s head’. Unless we make the assumption that explanations only occur under complete information (e.g. when both an antecedent event and its causal consequent event are present in working memory), a comprehension system requires some way of making a prediction to allow these statements to be connected.

Literary texts

Certain types of literary text rely on the comprehender’s construction of predictive inferences for their effects. Narratives with flashbacks, for example, encourage the comprehender to predict possible consequences of the flashback, and find inferential chains between those predictions and the current circumstances of the narrative [Zwaan, 1996]. In the relatively straightforward, linear narratives commonly employed in reading comprehension studies, predictive inferences are generally unnecessary and not explicitly encouraged.

Other forms of elaborative processing may not be so obviously ‘predictive’ (in a temporal sense), such as occasions where a comprehender ‘predicts’ details of an event. For example, if an agent in a text stirs a cup of coffee, the comprehender may infer that a spoon was used as the instrument of the event. These and related forms of temporally-concurrent inference, such as inferences about characters’ emotions, are also important in literary comprehension, where the comprehender employs a mode of processing which facilitates their involvement and helps them ‘explore the stimulus’ offered by the text [Zwaan and Graesser, 1993]. The focus on ‘non-literary’ texts in cognitive psychology has led to an under-estimation of the importance of elaborative processing in comprehension.

Computational efficiency

The third reason is a slightly more artificial and pragmatic one: in a comprehension system which can only form explanations by backward-chaining, some obvious and important inferences may be missed. To demonstrate this, consider the following observations:

$drive(agent:geoff, to:b, in:c), hold_up(agent:geoff, place:b)$

and the following set of rules:

- (1) $rob_plan(agent:X, place:Y) \longrightarrow$
 $go_step(agent:X, to:Y), hold_up_step(agent:X, place:Y).$
- (2) $go_step(agent:X, to:Y)$
 $\longrightarrow go(agent:X, to:Y).$
- (3) $drive(agent:X, to:Y)$
 $\longrightarrow go(agent:X, to:Y).$

Naturally, we would like the system to interpret the observations as ‘*geoff* drove to *b*, which explains how he went to *b*; this going event was a step in *geoff*’s plan for robbing *b*’. This is shown diagrammatically below:

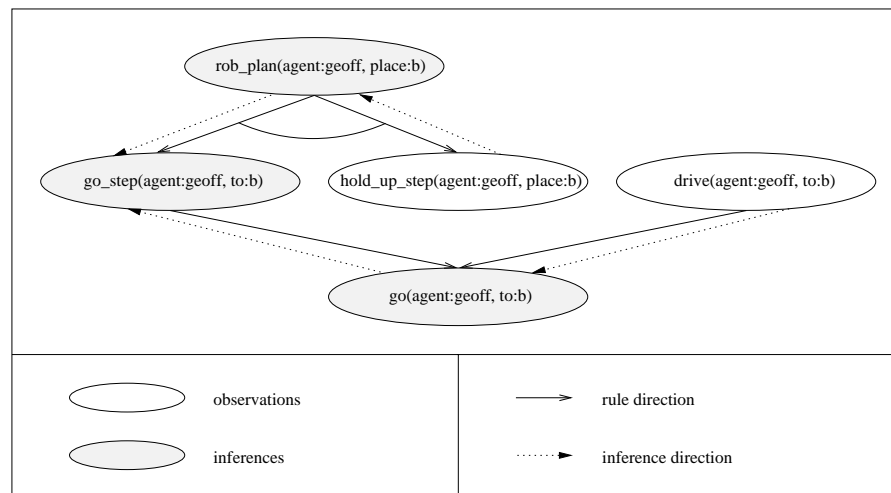


Figure 2.1: The necessity of bi-directional inference

Making the connection between $drive(agent:geoff, to:b)$ and $rob_plan(agent:geoff, place:b)$ requires at least one predictive inference: either from $drive$ to go (as shown on the diagram), or from go_step to go ; otherwise, the $drive$ event cannot be considered as being related to the rob_plan .

If bi-directional chaining is added to a comprehension system, most texts will consequently engender more inferences than if backward-chaining alone were allowed. For example, consider the above system’s interpretation of the text statement:

go_step(agent:paul, to:c)

If backward-chaining is the only mode of rule application, rule (1) alone is applicable (by matching the statement against one of its consequents).² If forward-chaining is additionally allowed, rule (2) is also applicable (by matching the text statement against its antecedent).

2.3.2 Incomplete Matching

A second method for increasing the power and flexibility of a production system is to allow *incomplete matches*. In strict rule-matching, only rules whose antecedent(s) (for forward-chaining) or consequent(s) (for backward-chaining) are completely matched may ‘fire’. This prevents errors being made; desirable for computational efficiency reasons, but not in cognitively-valid models. In the latter, it is essential to allow partial matches to account for certain psychological phenomena [Anderson, 1983]. In recent research, one text-related phenomena attributed to partial matching is the so-called ‘Moses illusion’. This effect occurs when ‘a term in a sentence or question (the “critical term”) is replaced with a semantically similar but incorrect term (the “distorted term”)’ and the comprehender responds ‘as if this distortion were not present’ [Kamas et al., 1996]. The effect gets its name from the archetypical example question:

‘How many animals of each kind did Moses take in the Ark?’

People will often respond to this question with the answer ‘Two’, even though Noah took the animals onto the Ark, and not Moses. This effect is often attributed to partial matching, where concepts in the question are matched against rules in memory which generate its semantic representation. Where there is sufficient match between the meanings of the critical term and the distorted term (e.g. both Moses and Noah are Old Testament characters), the ‘switch’ is not noticed; if there is insufficient overlap (e.g. Nixon is used as the distorted term), the distortion is recognised and no answer given.

Another perhaps related effect occurs during the reading of *role-shift* texts [Sanford and Garrod, 1981]. Here, comprehenders jump to conclusions about the roles of characters which later turn out to be incorrect. Their early assignation of a role to a character may be based on incomplete matches between rules in memory; once more information becomes available, matches with other role-assignment rules may be

²Incomplete matching would also be required, as described in section 2.3.2.

more complete, and the initial role assignment retracted as a result. The fact that such phenomena occur in comprehension points to the necessity of allowing incomplete matching in a comprehension system. This kind of ‘interpretation revision’ is central to my work and is discussed in greater detail in chapter 6.

Incomplete matching can contribute to inferential promiscuity by licensing inferences on the basis of very little information. Consider a system with the following rulebase (the system can only backward-chain on rules):

- (1) $hold_plan(planner:X, object:Y) \longrightarrow$
 $grasp(agent:X, object:Y).$
- (2) $juggle_plan(planner:X, object:Y) \longrightarrow$
 $skittles(Y), grasp(agent:X, object:Y).$

The system works by matching observations against the consequents of rules. Given the following text:

$grasp(agent:jill, object:s)$

both rules are applicable if incomplete matching is allowed. Even though the object of the *grasp* has no type defined, as indicated by rule (2), the inference to *juggle_plan* is allowed. If only complete matching were allowed, only rule (1) would be applicable; the missing information *skittles(b)* would prohibit application of rule (2), resulting in one less potential inference.

2.3.3 Number of Rules

Imagine we added two more rules to the system described in section 2.2 to yield the following rulebase:

- (1) $hold_plan(planner:X, object:Y) \longrightarrow$
 $grasp(agent:X, object:Y).$
- (2) $goal(planner:X, objective:(possess(agent:X, object:Y))) \longrightarrow$
 $hold_plan(planner:X, object:Y).$
- (3) $grasp(agent:X, object:Y) \longrightarrow$
 $throw_plan(planner:X, object:Y).$
- (4) $throw_plan(planner:X, object:Y) \longrightarrow$
 $goal(planner:X, objective:(dispose_of(agent:X, object:Y))).$

We could continue adding rules for relationships between actions, goals and plans in this manner, which would be absolutely necessary if the system were to cope with anything approaching a real text. Now, when the system comprehends the text:

grasp(agent:geoff, object:thing1)

even more inferences are produced: some of these are relevant and some less so. For example, the system could generate the possibly relevant inference ‘*geoff* has a plan to dispose of *thing1*’ by applying rules (3) and (4), rather than rules (1) and (2). The system could also create a contradiction by applying all four, resulting in ‘*geoff* plans to dispose of *thing1*’ and ‘*geoff* plans to hold *thing1*’.

Each time a rule is added, an interpretation containing the elements of the rule licenses more potential inferences. In addition, if several rules apply equally to the current interpretation and create multiple explanations for the same element(s), contradictions may arise.

2.3.4 Can Inferential Promiscuity be Prevented?

Each of the three factors above (bi-directional chaining, incomplete matching, and number of rules) increases the quantity of inferences which could potentially be generated during comprehension. When their effects are combined, this increase may be explosive, causing a system to grind to a halt under the processing load, or produce useless interpretations due to excessive, irrelevant detail.

However, each of the three elements is also necessary in a cognitive simulation of comprehension. Without these facilities, a system would suffer the following deficits:

- Without bi-directional chaining, certain types of texts may be incomprehensible or poorly-interpreted; computationally, the system may miss important inferences.
- Without incomplete matching, modelling of some psychological phenomena (e.g. human comprehension errors) is awkward. For example, inputs which *almost* match a rule may be completely ignored, when it would be preferable to infer the ‘missing’ part of the match.
- With few rules, the system’s area of application would be severely limited.

The answer is to factor in the facilities described above, but to constrain their application, hopefully to those occasions where it is actually warranted. In other words, there is a requirement for an *inferential control mechanism* which evaluates the inference process and decides which of the potential inferences to actually make, and/or which of the resulting information to incorporate into the evolving interpretation.

Control mechanisms are ‘fundamental to all cognitive processes and intelligent systems’ [Hayes-Roth, 1985]. Their importance to computational models cannot be overestimated: they both enhance their accuracy as models of cognition and their efficiency as computer programs. The specific control mechanisms at work during comprehension are a central concern of my work, and are outlined in the next section.

2.4 Inferential Control

For the purpose of explaining the types of control which could be applied to comprehension, I describe a simplified model of inference generation. This model has similarities to many previous ones (e.g. [Alterman, 1985], [Correia, 1980], [Thorndyke, 1976]), but at the same time has been generalised.

In this generic model, I assume that each interpretation derived by the comprehender is encoded into memory as one or more *representations*. Initially, representations only represent text statements; as more text statements are added to representations, the comprehender generates inferences to explain and elaborate them. Useful inferences can also be added to the representations. The items (text statements or inferences) manipulated during comprehension are called *representational elements*, or *r-elts*.

Figure 2.2 shows how the comprehender’s current set of representations may be extended to generate new representations.³

The diagram shows a single cycle of inference generation, starting from one set of representations and ending with a new set of representations. The processes (rectangles) shown in the diagram are as follows:

1. Match some or all of the current representations (*reprs*) to rules in long-term memory (similar to the example rules in the previous sections).
2. Apply the rules to the current representations to generate inferences (possible new r-elts).

³‘Representations’ is intended in the sense ‘one or more representations’.

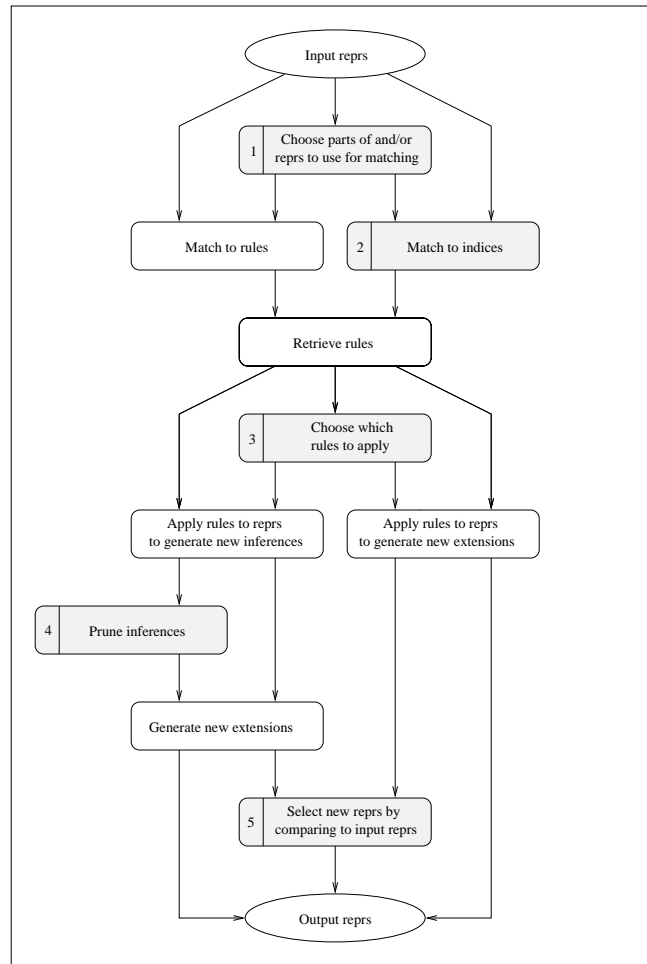


Figure 2.2: Generic process model for the inference cycle

3. Attach inferences to the current representations to produce new extensions. The term *extension* is used to denote the following:

- An existing representation, extended with some new inferences (or other alterations, such as deletions or transfers between memory stores (see section 6.4.1 on page 146)).
- Possibly some information about the quality of the resulting, extended representation (e.g. the probability that the new representation is correct, or a number representing the connectivity of the new representation).

Note that paths are possible through the diagram that avoid one or more steps. For

example, it is possible to generate extensions from current representations by applying retrieved rules directly: in this case, inferences are not generated as separate entities which are then attached to the representation. However, the alternative paths allow control to be applied at one or more points, as indicated by the shaded boxes on the diagram. The numbers in these boxes correspond to the following control mechanisms:

1. Focusing.
2. Directed rule retrieval.
3. Evaluation of rule matches.
4. Evaluation of inferences.
5. Evaluation of representations.

These mechanisms are described in the following sections.

Focusing

Before matching a representation to the rulebase, the comprehender may select a subset of the representation to be used for matching: for example, the most recently-encoded N elements of the representation. This restricted range of elements is usually referred to as *working memory* [Laird et al., 1987]; a more satisfactory term is *short-term store*, as ‘working memory’ is often defined as a cognitive system with both storage and processing potential [Baddeley, 1992].

In cognitively-motivated systems, the short-term store typically consists of the elements of a representation which reach some ‘activation threshold’ [Just and Carpenter, 1992]. Elements whose activation falls below the threshold are considered to have been encoded into the long-term store or forgotten; only those elements whose activation is above the threshold are used for matching against the rulebase.

Focusing reduces the gross number of rule matches which are possible, which can create considerable efficiency increases. Note that elements from long-term memory may still be retrieved into the short-term store, so that they can be incorporated into inferences [Trabasso and Magliano, 1996]. However, those elements which go ‘out of focus’ may be neglected, even when subsequent processing opens a route through which they could be integrated.

Directed rule retrieval

Rather than matching r-elts with the *whole* rulebase, the rulebase may be partitioned into sets of rules which apply under similar circumstances. Then, only those rule sets which are contextually appropriate to the r-elts may be considered as providing potential inferences.

For example, in Leake's ACCEPTER, the 'things to be explained' in a text are categorised as belonging to one or more *anomaly categories* [Leake, 1992]. Each rule (technically, *case*) in the knowledge base is also categorised according to the same anomaly vocabulary. In deciding which explanatory rules to invoke, ACCEPTER compares the text's anomaly categories with those of the rules in the rulebase; only matching rules are considered as potential explanations for the text. This 'directed' retrieval of applicable rules reduces search, as text elements do not have to be compared to every rule in the rulebase: only those rules indexed under the same anomaly category are considered as producing viable inferences. While assigning and matching anomaly categories may seem to involve its own overheads, these are outweighed by the efficiency benefits of less rule to r-elt matching.

Evaluation of rule matches

Once r-elts have been matched with the rulebase or to indices, there may be one or more rules which are applicable. Each applicable rule is a potential inference which the system could make. However, rather than making all of these inferences without further consideration, the system may select one or more which are most likely to yield useful extensions to representations. The decision about which potential inferences to select is often called *conflict resolution* [Charniak and McDermott, 1985]. Conflict resolution is usually carried out with respect to syntactic preferences, one of the most important being *specificity* (see section 5.1.1). Using this criterion, if one rule matched two r-elts and another only one, the rule which matched two r-elts is preferred; in other words, the first rule's match is more specific [Reichgelt, 1989]. Anderson cites specificity as important for dealing with exceptions to rules, such as irregular plurals in language production [Anderson, 1983].

Evaluation of inferences

Once inferences have been constructed, it is possible to evaluate the resulting output before it is accepted as an extension to a representation. Those inferences which do not add any useful information are ‘pruned’ out of the set of possible inferences; the remainder are then used to generate new r-elts which are attached to the current interpretation.

This form of control is used in Norvig’s FAUSTUS system [Norvig, 1989]. FAUSTUS suggests inferences using a weak marker-passing mechanism, which operates using a variant of spreading-activation (see section 4.3.1). Using the path taken by markers across a semantic network, FAUSTUS decides which paths could add useful information to the representation and which probably won’t. Those paths deemed possibly useful are then used to construct new representational elements; the remaining paths are discarded.

Other models which use relaxation in connectionist networks are employing a similar principle; this seems to be the case in Kintsch’s work, for instance [Kintsch, 1988]. Activation is passed between concepts, in much the same way as markers, and those concepts which receive sufficient activation are considered to be part of the representation. This topic is discussed in more detail in section 3.2.5.

Evaluation of representations

After inferences have been added to an interpretation, the representations constituting that interpretation are themselves evaluated. Those which are of the best ‘quality’ are retained, while the others are discarded.

For example, an interpretation may contain three competing representations. During the next inference-generation cycle, three inferences are made, one for each representation. These inferences are then attached to their respective representations, creating three new representations. The comprehender then evaluates the new representations, and finds that one is of significantly lower quality than the others. At this point, the comprehender may discard the low-quality representation, leaving the other two new representations.

Various metrics may be used to evaluate representations and assign quality ratings to them, such as Ng’s coherence metric [Ng and Mooney, 1990] or a simplicity metric [Thagard, 1989]. The evaluated representations may then be sorted according to their quality, and those below a certain threshold discarded; Ng’s ACCEL system uses beam-search to do this [Ng and Mooney, 1990].

(This description is brief as the concepts behind evaluation metrics and their use are discussed more thoroughly in chapter 5.)

Meta-level Monitoring

One additional form of control cannot be drawn on the diagram, as it operates *between* processing cycles, not within them. This form of control could be called *meta-level monitoring*.

Meta-level monitoring controls the other forms of control: it decides whether new inference-generation cycles should be instigated at all, by monitoring the current state of the comprehender's interpretation. This meta-level control mechanism thus continually tracks the evolving interpretation, deciding whether it is acceptable or requires improvement.

Interestingly, meta-level monitoring processes have received little attention in the AI literature. Comprehension is often assumed to continue until a structure at a pre-specified level of detail is instantiated. The reason for this may be that these systems often rely on a 'most-specific' definition of interpretation, inherited from diagnostic models, e.g. finding one of the set of 'allowable' hypotheses. For example, in Kautz's comprehension (plan recognition) system, the system does not halt until it discovers the minimal number of high-level plans that can be inferred from lower-level actions [Kautz and Allen, 1986]. This is not to say that these systems are bad, just that their behaviour can seem 'flat': each time a story is comprehended, the interpretation contains the same densities and types of information [Kayser and Coulon, 1981]; the interpretation also tends to be derived by the same process, over the same period of time, on every occasion the system is run. For example, if a system is based on scripts (see section 4.2), the triggering and instantiation of scripts occurs at a consistent pace: an initial event triggers a script, and subsequent events are used to fill the slots of the script. There is no method for slowing the rate at which scripts are triggered (for example, if there is little evidence to suggest they will be helpful); and no method for varying the amount of information which can be considered simultaneously (which may be useful if modelling short-term memory differences).

Evaluation of representations can give the appearance of meta-level monitoring in a computational system. Ng's ACCEL system, for example, constructs its interpretations to various levels of specificity, depending on whether an inference will add to their quality; once the best interpretation has been derived (i.e. no inferences can improve its quality), comprehension halts [Ng and Mooney, 1990]. However, this form of control is objective, as the criteria for quality are the same each time the program is run and the 'most-coherent' interpretation will always be derived.

The important point about meta-level monitoring is the idea of *acceptability*. This

is distinct from the goal to ‘find the highest quality interpretation’; instead, the goal is to ‘find an acceptable interpretation’, which may or may not be of the highest quality according to abstract, absolute criteria such as simplicity (see chapter 5).⁴ Acceptability is defined with respect to subjective criteria; these criteria differ between individual comprehenders, and may vary from ‘find the best possible interpretation’ (as in the most-specific interpretation) to ‘infer the few most important concepts’ (as in comprehension for ‘gist’).

Acceptability thus seems to require some notion of *representation quality* which would allow interpretations (and their component representations) to be rated, alongside a notion of *quality threshold* against which interpretations are compared. Once the quality of an interpretation reaches this threshold, an acceptable interpretation has been determined and comprehension can cease.

A detailed discussion of ‘quality’ is deferred until chapter 5. For the moment, let it suffice that there is some mechanism by which the comprehender can rate and compare representations.

I am now in a position to relate this idea back to the forms of control which are vital in preventing inferential promiscuity while maintaining cognitively important abilities (see section 2.3.4). In particular, *representation quality* and *quality threshold* could be used in an implementation of control as follows:

- *Focusing* could be made dependent on the quality of representational elements. For example, those r-elts of the lowest quality take priority on each inference generation cycle. If quality of an r-elt were defined in terms of the number of relationships between that r-elt and others, the lowest-quality r-elts would be those which are least-connected to other r-elts, and would thus be given priority.
- *Evaluation of inferences* could be determined by measuring the quality improvement afforded by an inference: those inferences making no improvement, or those inferences making the least improvement, could be discarded.
- *Evaluation of representations* can be directly related to representation quality: those representations falling within a certain range of quality values could be maintained and the others discarded.

⁴The idea of acceptability tallies closely with that of *satisficing* in problem-solving [Simon and Kadane, 1975].

- *Meta-level monitoring* is probably the most interesting and most neglected form of control. It can be applied by comparing the quality of representations with the quality threshold: those representations which fall outside this threshold are either discarded, or an attempt made to improve them; representations which cross this threshold are considered acceptable, at which point a decision to halt comprehension may be made.

Most of the suggestions made above have been incorporated into my theory of comprehension, and implemented using a particular model of representation quality and associated quality threshold.⁵ Control occurs at multiple levels and allows the comprehender to gradually filter out useless inferences and representations, hopefully leading to more economical, yet still cognitively realistic, comprehension. Later chapters specify these layers of control and their implementation.

2.4.1 Chapter Summary

In this chapter, I've described the core issue of this thesis: *inferential control*. I've demonstrated that inferential promiscuity can be a problem for computational models of comprehension; I also showed that its causes must be retained, otherwise interesting psychological aspects may be lost. I then showed that inferential promiscuity could possibly be curtailed by applying inferential control mechanisms. In the previous section, I suggested that a measure of interpretation quality, in tandem with a quality threshold, could encapsulate control effectively.

However, before reviewing the role of interpretation metrics in greater detail, it is necessary to define the framework into which these metrics fit. As I mentioned previously, the use of metrics is centred on inference-generation cycles, where interpretations are successively refined by the comprehender. Metrics for interpretations depend on an intimate understanding of the nature of interpretations, their representations, and the processes by which they are refined. To address this, the next two chapters describe the structures and procedures which 'implement' these ideas and which are common to both computational and psychological models of comprehension.

⁵Note that two of the forms of control not mentioned above, *directed retrieval of rules* and *evaluation of rule matches*, are also part of my theory, but rely on criteria independent of the quality metric; see chapter 6.

Chapter 3

Episodic Representations and Inference

This chapter introduces theories of episodic representation and its relationship to inference. The main themes are:

1. How interpretation can be defined, and how this relates to theories of episodic representation.
2. Inference processes: what an inference is, how inferences are made, and how they contribute to representations.
3. The role of episodic representations in comprehension.

3.1 What's the Point of Comprehension?

This may seem like a strange question to ask: because comprehension is central to many activities which we carry out on a day-to-day basis, we tend to forget that we are even doing it. One answer might then be 'Because we are human.' However, by trying to answer this question more precisely, we can get a handle on why people comprehend texts, as opposed to doing anything else with them. This may also give us an idea of how to apply control in comprehension: if we know why people halt their comprehension, settling for a particular interpretation, we may use this to shed light on control of computational models.

To answer the question, a first step is to define comprehension and look at alternatives to it: what can we do with a text if we don't comprehend it? I define comprehension as the construction of an interpretation which contains more information than that explicitly given in the text. In other words, an interpretation does not merely contain the *data* in a text, but contains the *information* derived from the data in the text; in the words of Dretske, an input containing many 'information-bearing' components (the data) is represented by fewer, less-informative components [Dretske, 1981]. This emphasises the constructive processes of comprehension, beyond simple representation of textual data.

An alternative to comprehending a text (in the sense of inferring information from its data) would be to represent with equal weight every datum in the text. The result of interacting with a text would then be a 'copy' of the text in memory (even the most mechanical comprehender, learning a text by rote, produces a richer representation than this).

There are several benefits to comprehending a text, over simply representing it in memory. The construction of text interpretations can be likened to a process of categorisation, where parts of the text are linked in memory into larger cognitive units [Barsalou and Sewell, 1985]. Like categorisation, this has certain benefits for the comprehender [Rosch, 1978]:

- *Economy*

A comprehender can't store every individual piece of data without generalisation, as this would require greater memory storage than they have available. Instead of representing new data individually in memory, a comprehender can 'summarise' those data by attaching them to an existing, structured concept. The more specific the data considered, the more specific the structure which can be applied and the less ambiguity there is in the interpretation.

- *Communication*

If culturally-shared structures ('frames of reference') are used to represent texts, interpretations can be shared with those who share those structures. By contrast, if everyone had a distinct interpretation of a set of data (e.g. a sequence of events occurring in a text), there would be no way for them to share information about those data. For example, if my conception of 'writing a program' were represented with unique structures that had no connection with cultural norms, there would be no way for me to share my programming experience with other people (e.g. students I'm teaching).

- *Learning*

By structuring the information in an interpretation and associating it with other parts of memory, a comprehender may learn new cognitive structures. While these structures are based on existing ones, they may contain insights which improve the comprehender's understanding of the world around them [Mooney, 1990]. Here, I don't mean that they can only learn about the mating habits of mosquitoes or the densities of stars, for instance; rather, they may learn more broadly, about how to interact with other people or even themselves, which in turn increases their chances of 'survival' (in the broadest sense).

If text data are stored as an undifferentiated mass, there is no way to relate the text to what the comprehender already knows. This then precludes the formation of cognitive structures: learning depends on processes such as analogy and induction, which are impossible without recognition of relationships between fragments of information [Kaplan and Berry-Rogghe, 1991].

By comprehending a text, it may be stored economically and in a format where it can be communicated to other people. In addition, it may be used as a basis for learning new cognitive structures which increase the comprehender's chances of functioning effectively in the world.

The most important of these issues as far as my own work is concerned is that of *economy*, or how the comprehender structures text interpretations. Given that there are many possible interpretations which could be generated (see section 2.3), it is important to define how these possibilities are explored, accepted and rejected, leaving a final interpretation. One way to think of this is as a process of reducing the number of possible 'text worlds' spawned by a text, to leave a single text world [Enkvist, 1989]:

...when we are exposed to an emerging text, certain elements and their collocations in the text activate references to a semantic universe of discourse, definable as a conceptually organised and retrievable system of models of reality, and lead us to a specific text world characterised by a highly constrained, specific set of states of affairs.

If we equate 'text worlds' with representations in memory, and the 'semantic universe of discourse' with the comprehender's knowledge base, the resulting 'specific text world' can be seen as the comprehender's final interpretation. We can begin to formalise this reduction process with Hobbs' pseudo-equation for comprehension [Hobbs, 1990]:

$$F(K, T) = I$$

where F is the comprehension process, taking as inputs the comprehender's knowledge base K and a text T . The output of this process is an interpretation, I . The important aspects of this formula are its emphasis on the combined role of knowledge and the text in *determining* the interpretation: there is no interpretation 'hidden' in the text which is wheedled out, but a dynamic process of construction and selection of representations, eventually leading to the comprehender settling on a set of the possible representations. This final set of representations constitutes their interpretation of the text.

The formula above expresses a relationship between the inputs and outputs of comprehension, but doesn't specify how representations are constructed and selected. Specifying these processes requires a model of how representations are constructed from input data; implementing these processes requires specifications which are computable. For this purpose, I next define how texts (T), representations (K and I) and comprehension processes (F) can be described computationally.

3.2 Computing Comprehension

In this section, I use deductive-nomological explanation as the foundation for a definition of comprehension [Hempel, 1966]; this model underlies computational work on abduction [Charniak and Shimony, 1994], which I describe in detail later in the section (expanding on section 2.3.1).

The general form of a deductive-nomological argument is [Hempel, 1966]:

$$\frac{l_1, l_2, \dots, l_r}{a_1, a_2, \dots, a_k} \\ E$$

In this equation, l_1, l_2, \dots, l_r and a_1, a_2, \dots, a_k are called the *explanans* (the things doing the explaining), and E is the *explanandum* (the set of things which are explained). The explanans consists of two types of element: *laws* (l_1, l_2, \dots, l_r) and *assertions* (a_1, a_2, \dots, a_k). In the field of scientific explanation (where this formula was first defined), the scientist is assumed to take a set of assertions a_1, a_2, \dots, a_k , such as facts about the world, and a

phenomenon to be explained, E ; their aim is then to determine l_1, l_2, \dots, l_r which make the above formula true.¹

Although this provides the *basis* for theories of abduction, there is a crucial difference. In abduction, the laws are known and the aim is to derive the assertions which, together with the laws, will semantically entail the explanandum. In other words, the aim is to derive $A = a_1, a_2, \dots, a_k$, where $L = l_1, l_2, \dots, l_r$ and E are known, such that

$$A \cup L \models E \quad (3.1)$$

is true. A is determined by inference from E and L .

If the members of L are rules relating causes to effects of the form *cause* \longrightarrow *effect*₁, ..., *effect*_n, and E is a set of effects or symptoms of unknown causes, abduction can be further characterised as inference from effects to causes or ‘deduction in reverse’ [Charniak and McDermott, 1985].² The A of formula 3.1 thus consists of the set of explanations for elements of E which make the above formula true. Each explanation is generated by application of the standard abductive syllogism [Brewka et al., 1997]:

$$\frac{b, a \longrightarrow b}{a} \quad (3.2)$$

For example, if *wet_lawn* is to be explained ($wet_lawn \in E$), and L contains the rule $rain \longrightarrow wet_lawn$, the following inference can be made:

$$\frac{wet_lawn, rain \longrightarrow wet_lawn}{rain}$$

i.e. *rain* is inferred as a cause of *wet_lawn*. Note that the resulting inference of the explanation *rain* makes formula 3.1 true:

$$rain, (rain \longrightarrow wet_lawn) \models wet_lawn$$

It is important to note that abduction is not logically sound. Despite this, it can find instantiations of A which satisfy the criteria for a valid explanation.

¹This is an extremely simplified version of Hempel’s theory, and it has many opponents in philosophy of science, e.g. [van Fraassen, 1977]. However, as I show in the rest of this section, it provides a reasonable grounding for a computational theory of comprehension.

²This interpretation of abduction is largely derived from diagnosis systems, where a set of hypotheses and their effects (e.g. medical conditions and their symptoms) can be easily encoded [Peng and Reggia, 1990].

This may seem a long way from text comprehension. However, many language comprehension tasks, such as resolution of pragmatic ambiguity and plan recognition, are particularly amenable to abductive techniques [Goldman, 1990], [Hobbs et al., 1993], [Ng and Mooney, 1992]. This is because abductive reasoning is a formalisation of ‘reasoning to the best explanation’, and it is a small leap of faith from this to defining comprehension as ‘finding the best explanation for a text’. This perspective is backed up by much research which shows how explanation is the driving force behind comprehension: while comprehending a text, people look for answers to ‘Why?’ questions, as opposed to ‘What happens next?’, ‘How?’, ‘Where?’, or ‘When?’ questions [Graesser et al., 1994]. Empirical studies based on verbal protocols have also shown the predominance of explanatory inferences over elaborative or predictive ones [Trabasso and Magliano, 1996]. Researchers in psychology have even begun citing AI research into abduction in papers about comprehension [Noordman and Vonk, 1998]. However, despite this, I have reservations about abduction as a complete theory of comprehension (see section 3.2.2).

3.2.1 Formalising and Visualising Comprehension

Text comprehension can be formalised as an abductive task by making the following assumptions:

1. Texts can be represented in some kind of semantic formalism. The formalism chosen is usually a propositional representation [Goldman, 1990], [Ng and Mooney, 1992]; my work follows this tradition, being generally classifiable as symbolic AI [Newell and Simon, 1972]. There is also significant evidence that propositions are valid units of psychological processing [Kintsch, 1998].
2. Propositions in the text are called *observations*. The set of observations comprising the text is designated by T .
3. Propositions can be manipulated by rules to generate new propositions and/or explanatory relations (I have already been using this assumption throughout the previous chapters). The set of rules is called a knowledge base and is denoted by K .
4. Observations and inferred propositions are incorporated into the comprehender’s representation R as *nodes*. It is important to distinguish between nodes, which represent propositions in memory, and the propositions from which they are derived.

The latter are used in tandem with explanatory relationships in deciding whether a particular set of propositions satisfies the abductive criteria of formula 3.1.

5. An explanatory relation can be constructed if a proposition B (either derived from the text or a previous inference) is in R and there is some rule $A \rightarrow B$ in K . In this case, the comprehender infers $A \rightarrow B$ by *copying* a relationship from semantic memory into episodic memory (see chapter 4). The comprehender also infers A as support for the relation. Note that this is *inference by abduction*, reasoning from effects to causes.

The final representation R therefore consists of the following elements:

- Observations represented as nodes.
- Inferred propositions represented as nodes.
- Inferred explanatory relations represented as arcs.

Returning to the simple example from chapter 2, we could use abduction to infer an explanation for someone's plan as follows:

Proposition from text:

$take_plan(planner:emma, object:orange1).$

Rule:

$goal(planner:X, objective:(possess(agent:X, object:Y))) \rightarrow$
 $take_plan(planner:X, object:Y).$

Inferred proposition:

$goal(planner:emma, objective:(possess(agent:emma, object:orange1))).$

Inferred relationship:

$goal(planner:emma, objective:(possess(agent:emma, object:orange1)))$
 $\rightarrow take_plan(planner:emma, object:orange1).$

This inference is based on syllogism 3.2. Note also that formula 3.1 is satisfied as the inferred relationship and proposition semantically entail the observation, i.e.

$goal(planner:emma, objective:(possess(agent:emma, object:orange1))),$
 $goal(planner:emma, objective:(possess(agent:emma, object:orange1)))$
 $\models take_plan(planner:emma, object:orange1).$

I have slightly changed the criteria of formula 3.1 by requiring that the representation *alone* explains the text, rather than a set of assertions and a knowledge base. This is done by copying relationships from the knowledge base into the representation, so that the representation becomes a valid abductive interpretation in isolation from the knowledge base.

The resulting representation can be visualised as an *episodic network*, similar to those used by psychologists [Kintsch, 1998], [Trabasso and Magliano, 1996], [van den Broek and Lorch, 1993]. The above representation can be converted into a network with two nodes, one for each proposition, and a single arc representing the explanatory relationship between them, as shown in figure 3.1.

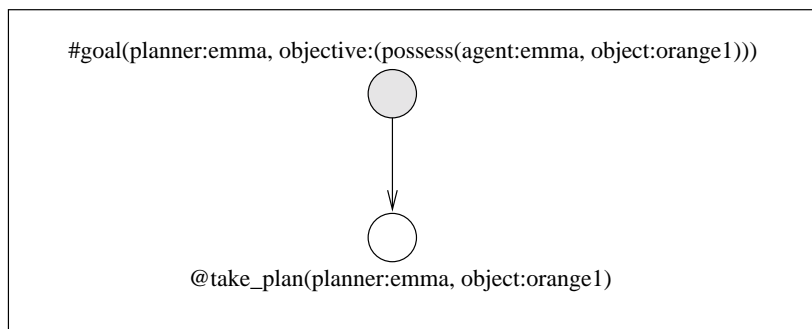


Figure 3.1: Episodic network representation of an abductive explanation

Also note that the nodes in the network have been marked with their origins:

- @ = nodes derived from observations.
- # = nodes derived by inference. These nodes are also shaded to make the distinction clearer.

This section has demonstrated how the criteria and syllogism for abductive explanation can be applied to interpretations of texts. However, there are important questions which are not answered by the basic model of abduction:

- *Are interpretations just explanations?*, i.e. can abduction deal with other types of non-explanatory relations?
- *Can observations explain each other?*, or must all explanatory relationships be constructed from inferred explanations to observations.

- *Should all observations be explained?* Are there any occasions on which it is better not to explain a text?
- *Are multiple representations maintained?* If comprehenders maintain multiple representations, can abduction model this?

In the next four sections, I show how these questions may be answered.

3.2.2 Are Interpretations Just Explanations?

If a model of comprehension is based entirely on abduction, this prioritises explanation to the exclusion of other kinds of inference. In particular, it prioritises explanation on the basis of logical implication. As I've stated before, explanation is by far the most important form of inference. However, other forms of inference do play a role: up to 30% of inferences made by readers of narratives are non-explanatory [Trabasso and Magliano, 1996]. In section 2.3.1, I described why it is necessary for a comprehension system to be able to make inferences by forward-chaining on schemas, which allows predictive and other types of inference to be made. In that section, I deliberately avoided technical detail for the sake of clarity. Now, with the technical background of section 3.2, it is possible to show more precisely how non-explanatory inferences relate to abduction.

What are these other forms of inference? This is a complex and confusing question to answer, because different researchers make different assumptions about the meanings of words like 'explanatory', 'elaborative' and 'predictive'. For example, one definition of 'elaborative' contrasts it with 'bridging', stating that elaborative inferences are not necessary for comprehension while bridging inferences are [Singer, 1994]; another definition claims that predictive inferences, possibly unnecessary for comprehension in most circumstances, can have an explanatory function in some situations [Murray et al., 1993]. Another example: some researchers differentiate between predictive and elaborative inferences [Whitney et al., 1991], while others treat predictive inferences as a subset of the class of elaborative inferences [Keefe and McDaniel, 1993].

To circumvent this confusion, I consider representations to consist of propositions and relationships between them, as in section 3.2. All relationships are directional, that is, they point from one proposition (the *parent*) to one or more other propositions (the *children*). Each relationship is treated as specifying a 'connection': the parent is a point of connection for the children; or, the parent is a more specific way of describing, classifying,

or linking the children. The structure formed by a parent and its children is termed a *tree*. An example in network form is shown in figure 3.2.

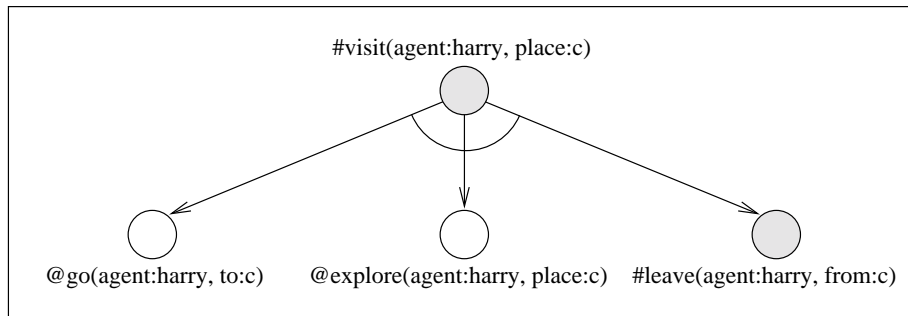


Figure 3.2: A tree representing a script-like explanation

The tree in figure 3.2 shows a relationship between a proposition representing a script-level event and several propositions at a lower level, representing the subevents of the script [Read, 1987]. (This tree is based on the schema described in section 4.2.1.) Note that both the parent and one of the children are inferred (marked with #). Also note that the relationship depicted is more complex than the one-to-one relationships previously described: this relationship connects a single parent to several children (one-to-many). The ability to create one-to-many connections is significant, for the reasons described in section 4.2.2.

I am now in a position to relate this simplified view to notions of explanation and elaboration. Each tree represents a relationship between the parent propositions and its children, providing a rationale for their co-occurrence in the representation. Trees may be generated either by forward- or backward-chaining, depending on which propositions are present in the representation when the inference is made.

The distinction between an elaboration and an explanation is in terms of the utility of a tree. If a tree's children have no basis in the text (i.e. all of its children are inferred and marked with #), and the children are not parents of any trees themselves, then the children *elaborate* the parent, and the tree is classed as *elaborative*. This functional view of representational elements specifies exactly when inferences are elaborative: when the resulting tree does not contain any children which are parents of trees, i.e. when its children have no explanatory function. An example elaborative tree is shown in figure 3.3.

Note that elaborations don't invalidate formula 3.1. The explanatory relationship between inferred propositions and observations still holds, but the formula is altered to

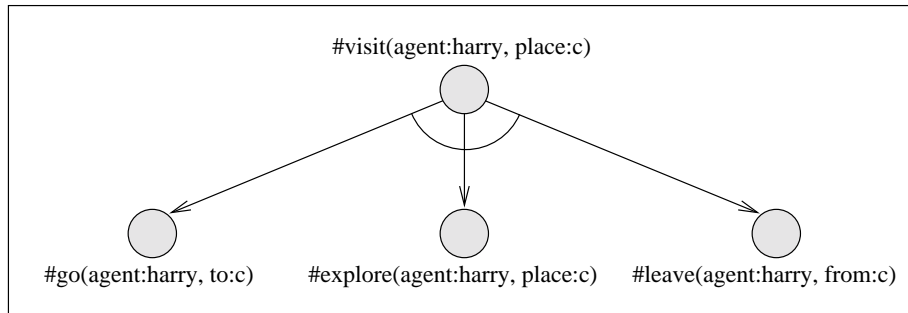


Figure 3.3: An elaborative tree

discriminate between propositions which explain observations and propositions which are just inferred children. The revised formula which takes account of this is:

$$X \cup E \cup C \models T \quad (3.3)$$

where:

- X is the set of inferred propositions which are parents (explanations).
- E is the set of inferred propositions which are children and not parents (elaborations).
- C is the set of relations, described as trees.
- T is the set of observations.

In this case, the presence of E makes no difference to the truth of the formula, as the propositions in E do not explain any of the propositions in T ; another way of looking at this is that they do not occur as parents in C . While this may make elaborative trees seem redundant, they are significant in terms of the interpretation metric I describe in chapter 5.

The other types of inference commonly referred to in the literature can also be incorporated into this framework, as described in the following sections. While some appear to be obviously explanatory (such as causal inferences) and others obviously elaborative (e.g. instrumental inferences), I attempt to show that the functional significance of an inference is more important than the superficial classification which it may be given. All types of inference may be either elaborative or explanatory, depending on their contribution to the quality of an interpretation.

Causal Inferences

Causal antecedent inferences establish sufficient causal explanation for an event [Graesser et al., 1994], [van den Broek et al., 1995]. For example, ‘John struck a match’ is a sufficient condition for ‘John lit his cigarette’. Given a node ‘John lit his cigarette’, a proposition representing ‘John struck a match’ may either be retrieved from the current representation or a new proposition may be created.

The difference between causal antecedent inferences and causal consequent inferences (the latter are often called predictive inferences [St. George and Kutas, 1998]) is the context in which the inference is made. For instance, if a comprehender reads a text containing the two ‘cigarette-lighting’ events above, there are two ways in which they could establish a causal relationship, as shown in figure 3.4 (next page).




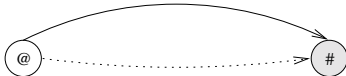
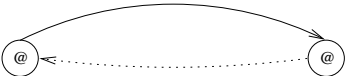

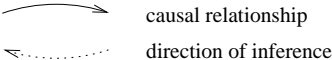
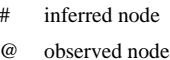
In the causal antecedent condition, the inference is backward in narrative time, from the lighting of the cigarette to the striking of the match. In the causal consequent condition, a prediction of a cigarette lighting event is inferred on the basis of a match being struck; this prediction is then *instantiated* when the lighting event occurs in the text [DeJong, 1979]. Note that in this case one proposition from the text is providing a causal explanation for another. The criteria for abductive explanation in formula 3.3 do not allow this, but I suggest an extension to amend this in section 3.2.3.

Causal antecedent inferences are one type of the broader class of *bridging inferences* [Singer, 1994]. Bridging inferences are considered by some to consist of inferences necessary for comprehension, as opposed to inferences which are purely elaborative (*ibid.*). This is the definition of bridging which I use here: any inference which contributes to explanation (as defined broadly in section 3.2) is considered a bridging inference. As such, causal inferences *may* be bridging inferences, or they may be elaborative, depending on their function in the interpretation.

Associative Inferences

I’m using the term ‘associative’ here in the sense of [Trabasso and Magliano, 1996]: inferences from a proposition which refer to information neither temporally consequent nor temporally antecedent to that proposition. Such inferences generally involve adding information about ‘features, properties, relations, and functions of persons, objects or concepts’ (*ibid.*). This information is temporally concurrent with the source proposition: for example, an inference from ‘Ted is a dog’ to ‘Ted is a mammal’ is not a causal antecedent or consequent inference, as Ted’s being a dog and a mammal are temporally

Figure 3.4: Causal antecedent vs. causal consequent inferences

Causal antecedent inference	Causal consequent inference
<p>Cycle 1</p>  <p>@struck(agent:john, object:match)</p> <p>Read in text proposition.</p>	<p>Cycle 1</p>  <p>@struck(agent:john, object:match)</p> <p>Read in text proposition.</p>
<p>Cycle 2</p>  <p>@struck(agent:john, object:match) @lit(agent:john, object:cigarette, inst:match)</p> <p>Read in text proposition.</p>	<p>Cycle 2</p>  <p>@struck(agent:john, object:match) #lit(agent:john, object:cigarette, inst:match)</p> <p>Infer consequent event (in this case, a prediction).</p>
<p>Cycle 3</p>  <p>@struck(agent:john, object:match) @lit(agent:john, object:cigarette, inst:match)</p> <p>Create causal relationship.</p>	<p>Cycle 3</p>  <p>@struck(agent:john, object:match) @lit(agent:john, object:cigarette, inst:match)</p> <p>Match inferred consequent event with next text proposition.</p>
	

concurrent. Some examples of associative inferences include inferences about instruments used in an event [O’Brien et al., 1988] and inferences about the spatial layout of locations in a narrative [Zwaan and Van Oostendoorp, 1993].

The structure generated by associative inferences is considered to be a tree expressing a connection between nodes, the same as other types of inference. Again, associative inferences may serve an explanatory or elaborative function [van den Broek, 1994]. For example, consider the following text:

Jane was reading a newspaper. A fly was annoying her. She swatted it.

The node representation of this text is shown in figure 3.5.

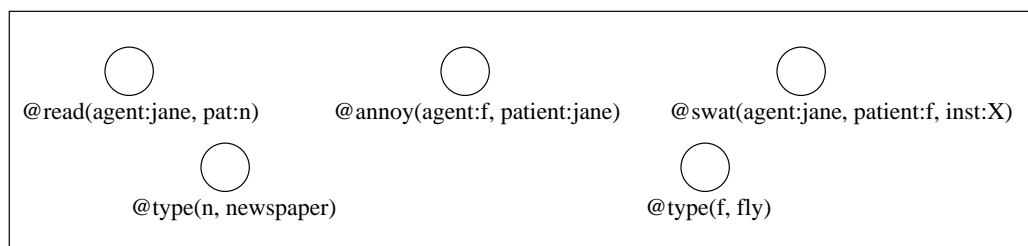


Figure 3.5: Representation of the ‘fly swatting’ text

Imagine that the comprehender has the following two rules:

- (1) $fly_swatting(agent:X, patient:Y, inst:Z) \longrightarrow$
 $swatted(agent:X, patient:Y, inst:Z), type(Y, fly),$
 $type(Z, swatter).$
- (2) $type(X, newspaper) \longrightarrow type(X, swatter).$

The comprehender is able to infer a *fly_swatting* on the basis of a *swat* event and an entity of type *fly*. However, this leaves the instrument of the ‘swatting’ unspecified. By inferring that a newspaper can be used as a swatter using rule (2), the comprehender can specify the instrument of the swatting event. This chain of inferences is shown in figure 3.6.

The inference from newspaper to swatter is associative (concurrent in narrative time, not a prediction, and not directly explanatory); however, it creates coherence by forming a connection between an explicit observation (the existence of a newspaper) and an inferred proposition which is part of the explanation of the swatting. Another way of stating this is that Jane’s possessing a newspaper ‘causes’ her to have a swatter to hand.

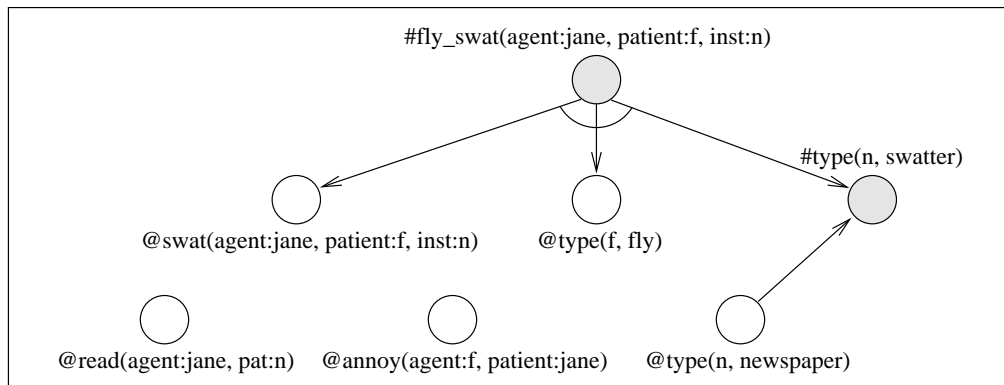


Figure 3.6: Inferences connecting newspapers and fly swatting

Although associative inferences seem to have little direct explanatory capability, they are important for instantiating roles within events, which can establish indirect explanatory connections between them. Other types of association, such as inferring the type of an instrument used in an event, may have a similar role in creating connections, for example:

Felicity drove the nail into the wood. Then she hit Mike.

Unlike the previous text, the type of the item being used to drive the nail is not mentioned. However, use of the word ‘nail’ may lead to an inference that a hammer is being used as an instrument. In turn, Felicity’s hitting Mike mentions no explicit instrument; however, if the existence of a hammer has been inferred, the availability of this in the representation makes it a likely candidate for this role.

Any type of inference may thus contribute to interpretation quality if it specifies relationships between elements of a representation. This formulation ties in with the idea of convergence and constraint satisfaction described in [Graesser et al., 1994]: an inference may be made when it receives ‘a high strength of activation from multiple information sources, and it satisfies the constraints from multiple information sources’. This is regardless of the direction of an inference (relative to narrative time) or the type of inference (causal, bridging, associative etc.). The important distinction is between explanatory inferences and elaborative ones; the time when an inference is made depends not on its type, but on whether making that inference will prove profitable at a certain point in comprehension [Noordman and Vonk, 1992], [van den Broek et al., 1995]. My computational interpretation of this idea is explored in depth in chapter 5.

3.2.3 Can Text Propositions Explain Each Other?

This question returns us to a consideration of how abduction can account for interpretations of texts. In formula 3.3, I showed how elaborative trees would have no impact on the truth of the formula, which still required a set of inferred parent nodes to explain the text. In other words, the observations stay on the right-hand side of the formula (they are the ‘things to be explained’) and the inferred explanations stay on the left-hand side.

However, in some circumstances, it may be necessary for observations themselves to provide explanations. This is often the case in AI models of abduction, such as ACCEL [Ng, 1992]. In one example from Ng’s thesis (*ibid.*), the following text is used:

Mary had a heart attack. John is depressed.

The explanation produced by ACCEL uses the observations $had(mary, h)$ and $heart_attack(h)$ to explain the observation $depressed(john)$, via the inferences: (1) if X has a heart attack, then X is in a bad condition; (2) if Y likes X , and X is in a bad condition, and X is irreplaceable, then Y is depressed. Again, formula 3.3 is satisfied by the propositions in the representation (adapted from Ng’s notation):

$$\begin{aligned} & @had(mary, h), @heart_attack(h), \#illness(h), \\ & \#bad_condition(mary), \#like(john, mary), \#irreplaceable(mary) \\ & [@heart_attack(h) \longrightarrow \#illness(h)], \\ & [@had(mary, h) \wedge \#illness(h) \longrightarrow \#bad_condition(mary)], \\ & [\#bad_condition(mary) \wedge \#like(john, mary) \wedge \#irreplaceable(mary) \\ & \longrightarrow @depressed(john)] \\ & \models @depressed(john), @had(mary, h), @heart_attack(h). \end{aligned}$$

(Note that the observations which are also explanations must appear on both sides of the formula.)

Is it important that observations are being used to explain each other, rather than inferred propositions? In a situation where two or more representations *could* satisfy the formula, priority should perhaps be given to those representations which contain the lowest number of inferred elements and hence the ‘smallest’ explanations (see sections 5.1.1 and 5.1.2). In other words, preference should be given to representations where paths of explanatory relationships originate and terminate at observations, over representations where inferred propositions are providing explanations. This is a version of the idea that some elements of the representation have priority over others, e.g. propositions

derived from observation or experiment [Thagard et al., inpr]. In the case of comprehension, observations have priority because they are *known to be true*; by contrast, inferred propositions always carry some degree of uncertainty. ACCEL relies on this for rating interpretations: the more paths there are between observations, or from intermediate nodes to pairs of observations, the higher the quality of an interpretation (see section 5.2.4 on page 103 for a fuller description of this metric).

By distinguishing between observations which are explained by inference and those explained by other observations, a system is not merely seeking to satisfy formula 3.3 but also trying to optimise *how* the formula is satisfied. This requires heuristic criteria which are difficult to define in terms of logical proof (see chapter 5).

3.2.4 Should All Observations be ‘Explained’?

Does every element of the text have to be ‘explained’ (or ‘proved’, in the abductive framework)? An alternative is to not explain observations, instead maintaining their ambiguity. Cases where ambiguity should be maintained (or further investigated) occur where there is insufficient evidence to point to a definite representation. For example, if a comprehender reads the sentence ‘John is depressed’, it is probably more efficient to attempt no explanation than to produce one of the many possible explanations for depression.³

A comprehender may also be unable to explain a phenomenon if they lack the necessary cognitive structures for its comprehension. In such cases, an attempt may be made to either find an analogical representation or generate a new structure which can account for the phenomenon (c.f. [Thagard, 1997]). In my model, I do not account for these creative mechanisms, concentrating instead on situations of the first type.

If no explanation is attempted of particular observations, the criteria for abduction (as defined in formula 3.3) are not met. Instead, we end up with the following:

$$X \cup E \cup N \cup C \models O \cup N \quad (3.4)$$

where:

³In safety critical situations, it may make more sense to attempt clarification of ambiguity rather than maintain it [Norvig and Wilensky, 1990]. During comprehension, attempts at clarification may take the form of actively seeking ‘missing information’, e.g. searching for more evidence to back up a potential interpretation.

- X is the set of propositions which are parents (explanations); these may be inferred or observed.
- E is the set of inferred propositions which are children and not parents (elaborations).
- C is the set of relations, described as trees.
- N is the set of observations which *are not* explained, i.e. observations which are neither parents nor children.
- O is the set of observations which *are* explained, i.e. observations which are children.
- O and N together comprise the text (T).

It seems as though the formula with which I began (formula 3.1) is becoming less and less adequate for deciding when a representation is satisfactory. The neat relationship between text propositions on the right-hand side of the formula and inferred explanations on the left is disappearing, as some observations provide explanations for others, and some observations are maintained without explanation.

While a valid representation may be considered primarily as an inferred set of explanatory propositions which must satisfy the formula, this gives no basis for deciding between competing representations. The problem comes from the formula's failure to recognise what is important about a representation: that is, the effort required to produce it (the number of inferred elements it requires) and the explanatory relationships it contains. The \models relationship doesn't really capture the nuances of the representation, instead describing it at a relatively abstract level, divorced from its *content*.

Some form of heuristic, such as a metric for interpretation quality (as suggested in the previous section), may be more effective in capturing these nuances. In my model, deciding whether or not to explain (or elaborate) an observation depends on the number of potential representations which could be generated from it: the greater the number of choices, the less impetus there is to generate one of those representations. However, the interaction between nodes which could be explained *within the same tree* causes the model to make a decision between those representations, discounting some of them and affirming others. This is described in detail in section 5.3.2 (page 110).

3.2.5 Are Multiple Representations Maintained?

A final issue is the extent to which competing interpretations and/or representations are maintained by comprehenders. In psychological research on syntactic ambiguity, it is often suggested that multiple representations are maintained until ambiguity is resolved [Just and Carpenter, 1992]. This is related to the issue of the previous section: an alternative to not explaining an observation (as in the previous section) is to generate and maintain all (or some) of the potential explanations, and collapse this set when some definitive evidence for one explanation over the others is encountered.

In comprehension at the level I am describing, there may be alternative representations for a text. As an illustration, consider a text consisting of a single sentence, ‘Mary was walking to school’, represented by proposition W . Imagine a comprehender who knows two possible explanations for this sentence: (1) Mary is walking to school because she is a schoolchild (S); (2) Mary is walking to school because she is a teacher (T). There are thus three possible explanations for the sentence; the propositional representations of these explanations satisfy formula 3.4:⁴

1. Mary is walking to school (i.e. no explanatory relations are created).

This ‘explanation’ can be expressed propositionally as

$$@W \models @W.$$

2. Mary’s being a schoolchild causes her to walk to school.

This explanation can be expressed propositionally (assuming causal rules) as

$$\#S, (\#S \longrightarrow @W) \models @W.$$

3. Mary’s being a teacher causes her to walk to school.

This explanation can be expressed propositionally as

$$\#T, (\#T \longrightarrow @W) \models @W.$$

The next question is how the comprehender stores one or more of these representations. There are three possible strategies which could be employed:

1. *Single best representation strategy*

Maintain a single representation which is of a higher quality than any other derived

⁴Note that there are no possible elaborations, as there are rules which would facilitate construction of a tree with only inferred children.

representation. If a single representation is maintained, conflicting information may force a re-evaluation of the representation and require extensive modification and deletion of information. In cases where the best representation turns out to be ‘correct’, this strategy should be most efficient, as it requires less storage of elements which eventually turn out to be irrelevant.

2. *Multiple representations strategy*

Maintain all competing representations as separate, discrete entities. By having separate representations available, a comprehender has the benefit of maintaining ambiguity: for example, if an action has two possible explanations, a comprehender may tentatively infer and store both; when further evidence supports one explanation over the other, the incorrect explanation may be deleted from memory. All representations may be maintained, or some criteria may be applied so that only the best subset of the representations is kept.

In cases where a text is misleading or difficult (either accidentally or deliberately), this strategy allows faster switching between representations: instead of deleting irrelevant segments of a representation and constructing new ones, a comprehender integrates information with representations which are best able to incorporate it. The disadvantage is that multiple representations have to be stored, which may be inefficient for easy and unambiguous texts.

3. *Partitioned relations strategy*

Maintain a single representation of the nodes, with multiple representations of relations between them. In this condition, the nodes common to all alternatives are stored without duplication. Each partition describes a set of relations connecting certain of the common nodes. Each set of relations and the nodes they connect is thus equivalent to a representation. (In some respects, this strategy is only an efficient method for implementing strategy (2).)

In each case, the one or more representations constitute an ‘interpretation’. Diagrammatic depictions of these strategies are shown in figure 3.7.

How can these strategies be realised computationally? In the next section, I describe one possible implementation technology.

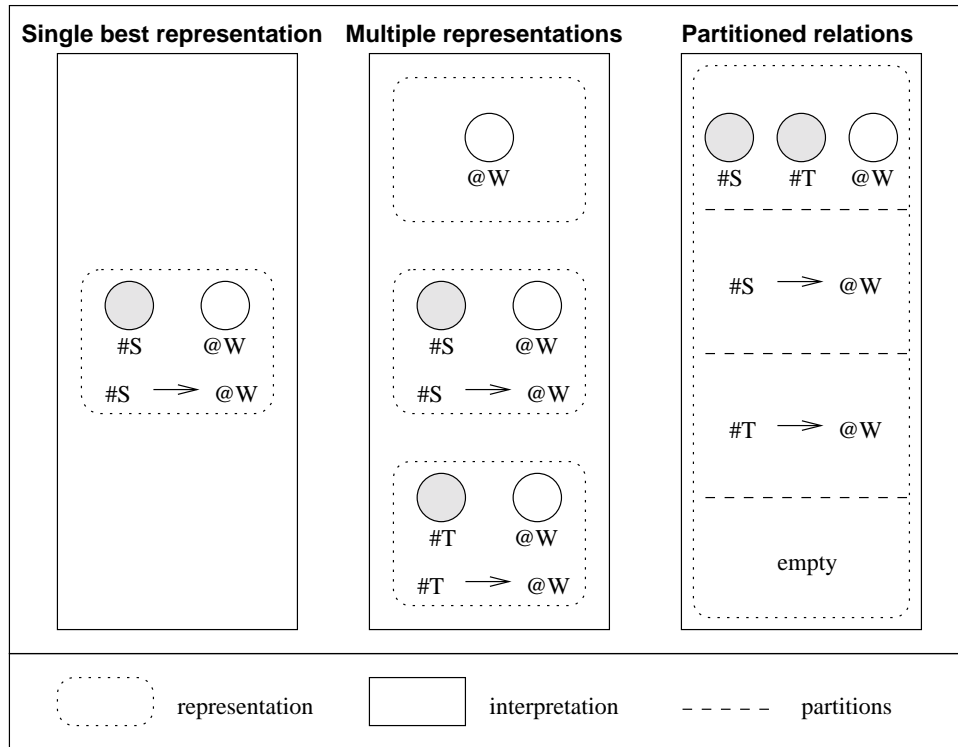


Figure 3.7: Possible structures for an interpretation

Assumption-Based Truth Maintenance

The *Assumption-Based Truth Maintenance System (ATMS)* provides support for multiple representations in an abductive context [de Kleer, 1986], [Ng, 1992]. In an ATMS, each node in the interpretation has a *label* describing the ways in which it could be explained abductively. A label consists of one or more *environments*, with each environment itself composed of *assumptions*. Assumptions are nodes which could explain the labelled node; that is, an environment for a node N must satisfy the familiar formula $E \cup K \models N$, where E is the environment and K is the knowledge base (see section 3.2). Additionally, other criteria may be enforced, such as requiring that an environment be minimal (i.e. not be a subset of another environment) and logically consistent [Brewka et al., 1997]. The format of a label is $\{E_1, \dots, E_j\}$, where E_1, \dots, E_j are environments. Each environment has the form $\{A_1, \dots, A_k\}$, where A_1, \dots, A_k are assumptions. An example labelling of an ATMS is shown in figure 3.8.

In a comprehension context, the aim is to find the labels for the nodes representing

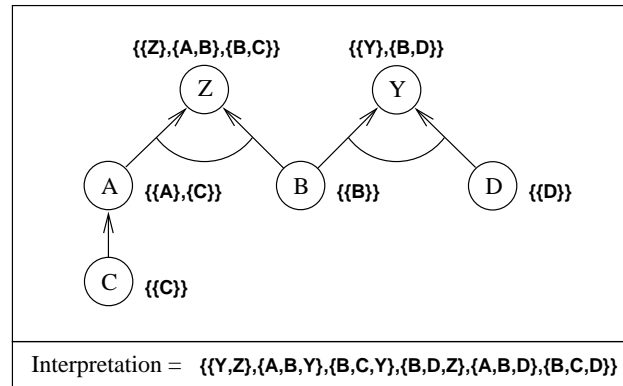


Figure 3.8: A labelling of an ATMS

the observations; the cross-product of these labels is the label for the whole text [Ng and Mooney, 1992] (cross-products are explained in a moment). As an example, consider a text consisting of two propositions, Y and Z . (I've dropped the @ and # symbols temporarily to aid exposition.) These are represented in the ATMS as two nodes, as shown in figure 3.8. The aim is thus to find a labelling for Y and Z ; the label for the whole text (its interpretation, in the sense of the previous section) is the cross-product of their labels.

The label of a node is a list of environments, and the first environment of any node is itself: in other words, for the node Z , the node may simply be assumed, creating the environment $\{Z\}$. Each of the other environments in a node's label are determined by backward chaining on rules which have that node as their consequent. The antecedents of the rule are a possible explanation for the node, so the cross-product of their labels is added to the existing label of Z . The cross-product of a pair of labels itself consists of environments; each of these new environments is produced by finding the set union of one old environment from each antecedent. For example, if the node being labelled is Z and there is a rule:

$$A \wedge B \longrightarrow Z$$

then the existing label of Z is appended to the cross-product of the labels of A and B . If A and B cannot themselves be explained, they have the labels $\{\{A\}\}$ and $\{\{B\}\}$ respectively (derived by the assumption that a node can always 'explain itself'). The cross-product of their labels is produced by finding the set union of pairs of environments from A and

B , such that each environment of A is crossed with each of B .⁵ For example, the only environment in the cross-product of the labels on A and $B = \{A\} \cup \{B\} = \{A, B\}$. This is added to the existing label of Z to give $\{\{Z\}, \{A, B\}\}$.

If there are rules which in turn provide possible explanations of an antecedent, the label on the antecedent is more complex. For example, if there is a rule

$$C \longrightarrow A$$

then the label on A becomes $\{\{C\}, \{A\}\}$, as C is a way of explaining A . This in turn means that the cross-product of the labels of A and $B = \{\{A, B\}, \{C, B\}\}$, and the label on $Z = \{\{Z\}, \{A, B\}, \{C, B\}\}$ (this is part of the example shown in figure 3.8). Labels are constructed by depth-first search, so that the root nodes of the ATMS are labelled first then the labels below generated recursively.

Having outlined the ATMS, I am now in a position to show how this corresponds with the possible structures for interpretations shown in figure 3.7. In the ATMS of figure 3.8, the cross product of the labels on the observations Y and Z is:

$$\{\{Y, Z\}, \{A, B, Y\}, \{B, C, Y\}, \{B, D, Z\}, \{A, B, D\}, \{B, C, D\}\}$$

This label is actually that of a ‘virtual node’ (V) added to the network by the ATMS, where V is equivalent to a node implied by the conjunction of the observations. For example, in this case, the virtual node V is implied by the conjunction of Y and Z ; in other words, a rule is added to the knowledge base of the form [Ng, 1992]:

$$Y \wedge Z \longrightarrow V$$

Each environment in the label for V is an abductive explanation for the text consisting of Y and Z , in accord with formula 3.1. To create a full representation (as defined in section 3.2), each environment could be extended with details of how it was derived and each of its nodes marked with its origin (i.e. inferred or observed). For example, the environment $\{A, B, Y\}$ is derived by using A and B to prove Z , so this relation is added to the environment; in addition, its nodes are marked appropriately. This gives:

⁵If there were three antecedents, each environment added to Z would be produced by finding the union of trios of environments, rather than pairs; and so on for four or more antecedents.

Nodes:

$@Y, @Z, \#A, \#B.$

Relations:

$(\#A, \#B) \longrightarrow @Z.$

Note that the ‘and’ (\wedge) between $\#A$ and $\#B$ has been replaced by a Prolog-style comma. This ‘extended environment’ can also be represented as a list:

$$[[@Y, @Z, \#A, \#B], [((\#A, \#B) \longrightarrow @Z)]] \quad (3.5)$$

To implement the strategies described on page 46, a system could be supplied with a heuristic for limiting the number of environments generated for each node. Ng employs this method in ACCEL, truncating the label on each node as it is generated [Ng, 1992]. This is done by sorting the environments according to a quality metric (see chapter 5), then keeping only the first n environments of the sorted list as the new label of the node. This method is basically a form of beam search, where n is the width of the beam.

To demonstrate this, consider a heuristic labelling of figure 3.8, based on sorting environments according to a quality metric. The quality metric used here is based on simplicity: quality is determined by dividing the number of observations explained by the number of assumptions in an environment [Ng, 1992]. For example, the above environment (labelled 3.5) has quality = $\frac{1}{3}$ ($@Z$ is explained, $@Y, \#A$ and $\#B$ are assumed). Each environment is annotated with its simplicity rating.

Now, to implement maintenance of a single representation, the label of each node is truncated so that only the highest quality environment is maintained. To implement maintenance of multiple representations, the n ($n > 1$) best environments are kept in each label (or one environment, if only one is derivable).

The table on the next page shows the difference between maintaining single or multiple representations. Each environment of figure 3.8 has been extended as suggested in formula 3.5 and annotated with its quality. The resulting format for an environment is [*Nodes*, *Relations*, *Quality*], where *Nodes* and *Relations* are lists and *Quality* a real number. Labelling is carried out as described previously. Environments discarded due to truncation of labels are marked with ‘*’ and are in italics; where there are two equally good environments, the one with fewest nodes is maintained and the other discarded.

Node	Label when n=1 (single representation)	Label when n=2 (multiple representations)
#C	[[#C],[,0]	[[#C],[,0]
#A	[[#A],[,0] *[[#A, #C],[(#C → #A)],0]	[[#A],[,0] [[#A, #C],[(#C → #A)],0]
#B	[[#B],[,0]	[[#B],[,0]
#D	[[#D],[,0]	[[#D],[,0]
@Y	[[#B, #D],[(#B, #D) → Y], $\frac{1}{2}$] *[[@Y],[,0]	[[#B, #D],[(#B, #D) → Y], $\frac{1}{2}$] [[@Y],[,0]
@Z	[[#A, #B, @Z],[(#A, #B) → @Z], $\frac{1}{2}$] *[[@Z],[,0]	[[#A, #B, @Z],[(#A, #B) → @Z], $\frac{1}{2}$] [[#A, #B, #C, @Z],[(#C → #A), (#A, #B) → @Z], $\frac{1}{2}$] *[[@Z],[,0]
Interp. (V)	[[#A, #B, #D, @Y, @Z],[(#A, #B) → @Z], (#B, #D) → @Y], $\frac{2}{3}$]	[[#A, #B, #D, @Y, @Z],[(#A, #B) → @Z], (#B, #D) → @Y], $\frac{2}{3}$] [[#A, #B, #C, #D, @Y, @Z], [(#C → #A), (#A, #B) → @Z], (#B, #D) → @Y], $\frac{2}{3}$] *[[#A, #B, #C, @Y, @Z], [(#C → #A), (#A, #B) → @Z], $\frac{1}{3}$] *[[#A, #B, @Y, @Z], [(#A, #B) → @Z], $\frac{1}{3}$]

Table contrasting maintenance of single and multiple (=2) representations.

ACCEL uses this method to control the construction of representations, storing a set number of environments on each label (including the label representing the final interpretation). It can thus simulate the single vs. multiple representations strategies. However, labels are truncated according to the number of environments they contain, rather than the quality of those environments; quality is merely used to sort the alternatives. If there are many environments close in quality, some may be discarded simply on the basis of quantity; if there are two environments, one of very high quality and one of low quality, both may be maintained. This problem is caused because the criteria for truncating labels are based on volume, rather than content, of labels.

In my own model, IDC, the mechanism for managing the interpretation is broadly based on an ATMS architecture, though this is not particularly explicit (see chapter 6 for details). More importantly, IDC incorporates a parameter which can be set to simulate strategy (1) or (2) (strategy (3) is not directly implemented, only simulated - see section 6.3.3 on page 143). However, decisions about which representations to maintain are based on representation quality: any representations which are significantly worse than the best representation may be removed from consideration. Because the decisions are based on quality differences (rather than maintenance of a fixed number of representations), in cases where quality differences are small, all of the alternatives may be maintained; and in cases where there are few representations with widely differing qualities, the poor quality representations may be removed regardless of how ‘full’ the stack is. The elements of my model which handle representation choice are described in greater detail in section 6.4.2 (page 6.4.2).

IDC also uses ideas from the ATMS in calculating the incoherence of nodes in the knowledge base. Very briefly, the knowledge base is labelled in an ATMS-like way; the occurrence of nodes in labels is then used to determine how often nodes are likely to occur in representations. This process is described in more detail in section 5.3.3 (page 114).

There is an alternative view of comprehension which is related to the question of multiple representations. This strand of research is currently important for text comprehension psychologists, and can be broadly classified as ‘symbolic-connectionist’: a hybrid theory whose models combine elements of propositional representation, production system architectures, and connectionist algorithms for constraint satisfaction [Holyoak, 1991]. Proponents of these models often criticise ‘pure symbolic’ models for the following reasons:

1. Their view of representations as discrete, distinct, symbolic structures in memory.

2. Their lack of flexibility in processing.

Point (2) is discussed in detail in chapter 4, and particularly in section 4.2.1.

I now discuss point (1), describing symbolic-connectionist theories of interpretation and how these relate to traditional symbolic models.

Symbolic-Connectionism and Interpretation

A chief aspect of the symbolic-connectionist approach to comprehension is the rejection of models which find the correct representation of a text using very ‘smart’ processes, such as models based on scripts or other schemas [Kintsch, 1988]. One problem with smart models is that very little irrelevant information is produced during comprehension: the representation evolves in an orderly, tidy fashion. This does not mean that the wrong representation is never produced; often, a drastically poor representation may be derived based on tiny amounts of irrelevant information. Rather, it is the fact that these models *only* find one representation to which they cling tenaciously, even in the face of contradictory or ambiguous information.

Kintsch contrasts this with a view of comprehension as the initial promiscuous construction of inferences, some of which may be incorrect, irrelevant and/or contradictory, followed by integration of this information into a stable representation (*ibid.*)⁶ Alternative explanations or elaborations may loiter in memory for a while, but can gradually fade out of the representation if they prove to be unhelpful in forming coherence. The resulting representation is a network of nodes with attached activation levels; van den Broek et al. suggest that the level of activation of a node corresponds with its recall probability, which in turn corresponds to the strength of its encoding [van den Broek et al., 1996].

A couple of issues are raised by this viewpoint:

- How is a stable representation in memory maintained, given that the representation for a text is characterised as a pattern of activation?
- How could *several* stable (though perhaps competing or contradictory) representations be maintained? For example, is there some way of indicating that one pattern of activation denotes one representation, while a second pattern denotes another?

⁶Note that this characterisation of the inference cycle can be mapped onto figure 2.2 of chapter 2, with the only controls applied being of type (3) (the rules applied are chosen probabilistically) and type (5) (a relaxation algorithm is used to settle on a new representation).

In answer to the first point, Kintsch states that episodic network representations of texts are separate from the semantic networks used to construct them [Kintsch, 1988] (I return to this distinction in chapter 4). In a sense, episodic representations are constructed by *copying* elements from semantic memory [Alterman and Bookman, 1992]. A (tiny) example is shown in figure 3.9.

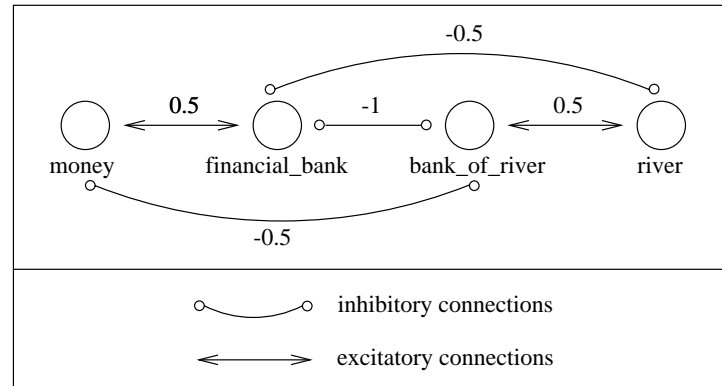


Figure 3.9: A representation of the meaning of ‘river bank’ (after [Kintsch, 1988])

This representation shows both nodes constructed as a result of encountering certain words in an input, and inferred associates of those nodes. Each node is assigned an initial activation value, according to whether it is part of the text or inferred. For example, if the comprehender encounters the phrase ‘river bank’, we could assume that *river* and both *financial_bank* and *bank_of_river* receive some activation. The network is then ‘relaxed’ by spreading activation between the nodes, according to the connections between them. This causes the activation of some nodes to drop to zero, while others remain activated (see [Kintsch, 1988] for full details of the algorithm). Those nodes which remain activated constitute the ‘meaning’ of the initial input. The program for performing relaxation of the network is relatively simple; my own implementation produces an ‘activation profile’ for the four propositions of figure 3.9 as shown in figure 3.10:⁷

As can be seen from this profile, the propositions *river* and *bank_of_river* remain active, while the activation of *financial_bank*, initially above zero, is quickly inhibited until it ‘drops out’ of the representation.

⁷My own implementation of it is written in SICStus Prolog and based on Kintsch’s presentation of the Construction-Integration model in [Kintsch, 1988]; technical information on matrix multiplication came from Pearl [Pearl, 1988]. I tested the implementation on Kintsch’s examples to ensure that the implementation was producing the correct results (it was).

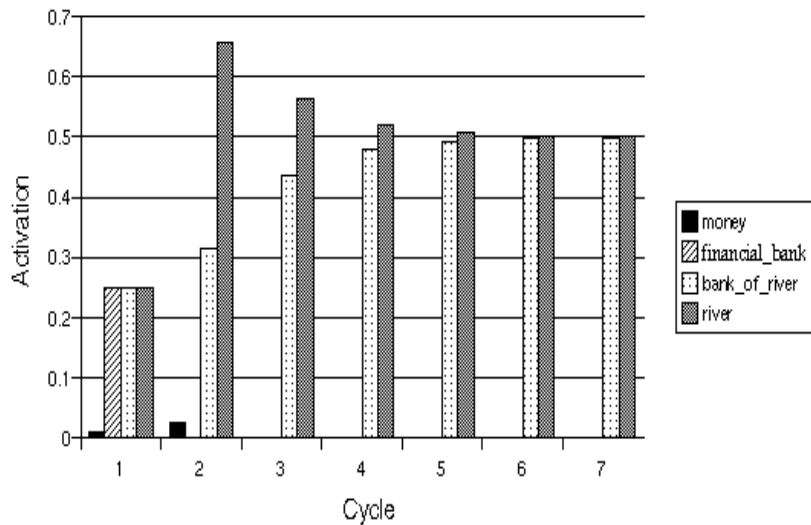


Figure 3.10: Activation profile for concepts involved in comprehending ‘river bank’

How does this relate to the issue of multiple representations? The activation profile demonstrates how simultaneous meanings for a word, phrase or other cognitive unit which are initially contradictory may be resolved to a single unambiguous representation. In this case, the meanings of bank as ‘financial bank’ and ‘bank of a river’ initially compete for activation; however, the extra activation afforded the latter meaning by the presence of the word ‘river’ eventually proves decisive. The final episodic representation is therefore a pattern of activation across a set of nodes; those nodes which carry no activation are ‘absent’ from the representation.

If the word ‘bank’ is encountered in the context ‘the bank collapsed’, the resulting activation profile might look like figure 3.11 (page 57). Here, the representation has not settled on a single meaning of the word ‘bank’, but instead maintains both alternatives. This is because the word ‘collapsed’ is equally connected to both the ‘financial institution’ and ‘river bank’ meanings for ‘bank’. Distribution of activation over the alternatives therefore corresponds to an ambiguous interpretation where alternative representations are implicitly maintained.

The relaxation process produces activation profiles which can be compared with psychological data (for example, Kintsch compares the Construction Integration model with reaction time data in [Kintsch, 1988]). However, while ‘relaxing’ representations show the time course of activation over concepts, there is no ‘meaning’ within those activations

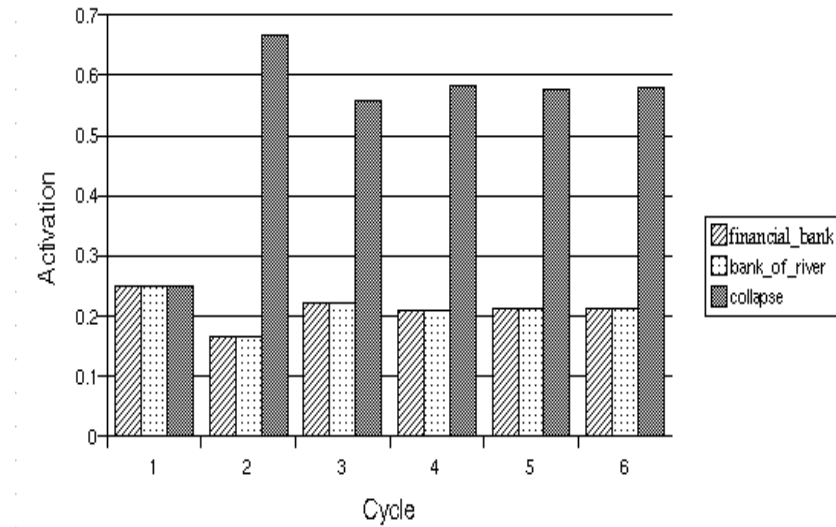


Figure 3.11: Activation profile for concepts involved in comprehending ‘the bank collapsed’

beyond ‘concept A is activated more strongly than concept C’. There is no content in the connections between concepts, and it is thus difficult for a report of comprehension to be extracted from them. This lack of content arises from the treatment of nodes as ‘associations’; there is no explicit process of *explanation*, and the processes for *reasoning* about texts are often absent. For example, Kintsch states that the basic construction-integration model provides no account of strategic inference processes, such as the construction of bridging inferences [Kintsch, 1988]:

...the generation of additional inferences [...] is necessary because not all inferences that are required for comprehension will, in general, be obtained by the random elaboration mechanism [*that is, the promiscuous construction phase described above*]. In some cases more focused problem-solving activity is necessary to generate the desired inferences. Exactly how this is to be done is, however, beyond the scope of this article. [Kintsch, 1988]

Garnham notes this tendency in [Garnham, 1996]:

...the relation of association is not sufficient to model links between pieces of information conveyed by texts. One piece of information may be strongly (or weakly) linked to other pieces in various ways.

He also notes that relations of different types cannot be expressed by associative links alone. (Although my own model uses a single kind of ‘link’, other types of relation can be expressed by treating them as nodes (see chapter 6).)

In contrast to symbolic-connectionist models, pure-symbolic models explicitly describe the processes which lead to a particular representation. In addition, the reasoning process which constructs episodic representations is intimately coupled with the mechanism for deciding which representations to maintain. This reasoning process is also implicitly encoded into the representation: concepts are explicitly ‘bundled’ into structures which reflect their derivation (c.f. phrase-structure trees or proofs). This gives far greater scope for retrieving something akin to a verbal protocol from such systems [Trabasso and Magliano, 1996].

3.2.6 Chapter Summary

In this chapter, I have begun to address some of the issues I introduced in chapter 2:

- I have shown how comprehension can be characterised computationally.
- I have defined how interpretations and representations relate to one another: interpretations can be described as sets of representations.
- I have established that representations may be constructed and manipulated computationally, and that the resulting representations broadly qualify as an interpretation.
- I have demonstrated that abductive explanation alone cannot be assumed to define a representation; instead, some form of heuristic is necessary to specify representation quality. The quality ratings can then in turn be used to manage the interpretation.

In chapter 4, I turn to definitions of semantic representations (abstract representations in long-term memory). This helps define more accurately the sources of episodic representations (i.e. representations which are part of an interpretation). This chapter and the next are thus complementary: the episodic representations I have been concerned with in this chapter are intricately connected to the semantic representations described in the next.

Chapter 4

Semantic Representations

This chapter introduces some central theories about semantic representations in computational models of comprehension. The main themes are:

1. The role of semantic representations in comprehension.
2. Misconceptions about schemas and the distinction between declarative and operational aspects of comprehension.
3. An analysis of associative networks as a representational scheme, and an argument for my use of structured schemas.

This chapter also continues the discussion of inference which underlies this thesis.

4.1 The Semantic/Episodic Distinction

The distinction between *episodic* and *semantic* representations was initially proposed by [Tulving, 1985]. A principle difference between these forms of memory is their dependence on *context* [McKoon et al., 1986]: episodic memory contains representations of specific events, places, people etc., at particular times and in particular places, and is thus context-dependent; semantic memory contains general abstractions from instances in episodic memory which are context-independent and may be culturally shared [Hintzman, 1986], [Tulving, 1986].

This is the standard paradigm for descriptions of memory in comprehension research. However, there have inevitably been attempts to show that there is really no distinction

between the two types of memory [McKoon et al., 1986]. As a small example: McKoon et al. performed an experiment using recognition and lexical decision, hypothesising that the former was a ‘prototypical’ task for episodic information, and the latter a ‘prototypical’ task for semantic information [McKoon and Ratcliff, 1979]. They argued that information from one memory store should not influence performance on the task which probes the other store. However, their experiments instead showed that episodic information led to priming in the lexical decision task, and semantic information led to priming in recognition.

The separation of semantic from episodic information is easier in computational models. This is because there is a sharper boundary between the rules which are used to manipulate data (*semantic memory content*) and the data themselves (*episodic memory content*). For example, in a language processing system, a structure derived by parsing a sentence resides in episodic memory, while the rules used to produce the parse reside in semantic memory.

Even here, though, recent research has sought to break down the boundary: for example, the content of episodic memory is not extensively abstracted to form semantic content (as in case-based reasoning [Kolodner, 1992]); or actual episodes may be used in place of rules (as in instance-based reasoning [Aha et al., 1991] or example-based parsing [Somers, 1992]). These approaches delay the need for abstraction until a pertinent situation arises.

The approach I take is traditional, in that I assume a separation between semantic and episodic memory. I assume representations in semantic memory which have been derived by some learning process (such as abstraction from representations in episodic memory [Mooney, 1990]). However, I do not concern myself with these learning processes.

The representations in episodic memory are constructed via inferences which employ schemas (see below) in semantic memory. The basic mechanism behind this is *copy-based* [Alterman and Bookman, 1992]. In effect, this means that the representations in episodic memory are formed by copying representations from semantic memory; the central difference is that the representations in episodic memory have *instantiated variables*.

The schema is a particularly important form of semantic representation. In chapter 2, I briefly introduced schemas and their computational manifestations. In the remainder of this section, I describe schemas in more detail, as they form the basis of the semantic knowledge representation of IDC. In particular, I describe the following aspects of schemas:

- Their general characteristics.

- Criticisms which have been directed at schemas. I show how these criticisms are based on a misleading conflation of ‘top-down’ processing with ‘predictive’ processing.
- By emphasising the declarative aspects of schemas, I can compare them to related representational formalisms, such as associative networks. I then show how associative networks and schemas can be uniformly represented.

4.1.1 Overview of Schemas

A *schema* is a structure in semantic memory which represents a ‘cluster of concepts’ [Eysenck and Keane, 1995]. Although schemas are interpreted differently according to the kinds of knowledge they are structuring, they have many common characteristics, as described below (extended and generalised from Minsky’s statements about frames in [Minsky, 1975]).

Schemas Support Interpretation through Inference

The forms of inference supported by schemas include both forward- and backward-chaining (see section 2.3.1). These can be implemented by assuming that a schema is equivalent to a logical implication, and is thus a valid basis for logical inference (see section 4.1.2).

Schemas are Networks within Networks...

Minsky states that ‘We can think of a frame as a network of nodes and relations’, obviously influenced by Quillian’s theory of semantic networks [Quillian, 1968]. Similarly, schemas can be conceived of as collections of nodes and relations; each node represents a concept, and relations show connections between related concepts. Relations may be typed, to represent different kinds of connection: *isa* relations (read as ‘is a’, as in ‘a dalmation is a dog’) and ‘has-part’ (as in ‘car has-part steering wheel’) are two examples. In addition, a schema for one type of situation may be linked to a schema for another type of situation. For example, the ‘living-room’ schema may be connected to the ‘house’ schema. So, a schema is a network of related elements which can itself be a node in a larger network; presumably, this larger network may be a node in another network, and so on...

The main difference between semantic networks and schemas is the bundle of links inherent in schemas. In a semantic network, relations between nodes are accessed one

at a time; in a schema, a bundle of links may be accessed en masse. This has led some researchers to criticise schemas as being ‘too top-down’; however, they seem to be confusing the declarative definition of scripts with their operational definition (see section 4.2.1).

Schemas Contain Slots

Once a schema has been activated in response to an input, the nodes attached to the schema become available for ‘filling’. These nodes are *slots*: conceptual spaces into which actual instances are inserted as they are encountered in the input. A slot can carry *constraints* which specify the range of values with which the slot may be filled; for example, a slot for ‘human height’ might carry a constraint like ‘greater than 30cm and less than 270cm’ (to accept both exceptionally short and tall people). A slot may also carry a *default* which is used if no information is available to the contrary: for example, the ‘human height’ slot may carry a default like ‘180cm’. Defaults may be overridden if actual values are specified.

Schemas Can Bypass Logic

Minsky intended his theory of frames to allow types of inference which were unacceptable in classical logic, such as making default assumptions based on incomplete evidence (see above). However, since publication of Minsky’s theory, logic has evolved to accommodate exactly these kinds of inference, e.g. Default Logic [Reiter, 1980] and Circumscription [McCarthy, 1980]. In addition, others have demonstrated the correlations between schemas and logic [Hayes, 1980]: as I show in section 4.1.2, both schemas and the forms of inference associated with them can be represented in logical form. In some respects, this makes redundant the claim that schemas can perform non-logical inference.

Schemas Can Be Implemented

Schemas are data-structures which can be implemented computationally in several ways, for example, as production rules or declarative formulae. In the former case, the schema contains information about how it is to be applied and how to construct representations; in the latter, schemas contain only descriptive information, without any indication of how it is to be applied. However they are implemented, perhaps the most important point is

that schemas can be represented with enough precision to be manipulated consistently by an algorithm.

As I have hinted here, schemas share similarities with several other representational formalisms. In particular, schemas may be expressed as pseudo-logical expressions, which can in turn be implemented in a computer language such as Prolog. The next section clarifies this as a background to my discussion of scripts and a justification of the representation of semantic information in IDC.

4.1.2 Schemas and Logic

One criticism faced by advocates of schema-based theories of representation, particularly in the field of artificial intelligence, is that schemas have no well-defined semantics or proper rules of inference [Bartsch, 1987]. In psychological accounts of schemas, there is no real need for either, as abstract descriptions of frame content and the processes are sufficient to make predictions about behaviour (e.g. Haberlandt and Bingham's anticipation of a processing advantage for events presented in 'script' order, as opposed to reverse order [Haberlandt and Bingham, 1984] - see section 4.2). In AI, though, systems with ad hoc representations quickly become very messy and complicated, with algorithms having to be tailored to the particular formalism [Norvig, 1989]. This is a problem in traditional production systems, where semantic knowledge is mixed with instructions which make changes to memory states [Laird et al., 1987].

An alternative is to represent semantic knowledge in some form which has a clear semantics and associated inference rules. First-order predicate calculus (FOPC) is an example of such a formalism. Although limited in its expressivity, it can easily encapsulate schema-type knowledge. These FOPC expressions contain no instructions about what to do with concepts, instead just stating the static semantic relationships between them. Several researchers have suggested how schemas may be represented in FOPC ([Hayes, 1980], [Hobbs et al., 1993], [Hongua, 1994]) and related formalisms such as Discourse Representation Theory ([Bartsch, 1987]). These suggestions can be merged to yield a useful notation for schemas, as follows:¹

$$\textit{Schema_concept} \longrightarrow \textit{Slot_concept}_1, \dots, \textit{Slot_concept}_n.$$

¹This is the notation I introduced briefly in section 2.2.1.

I assume that schemas express a relationship between a concept at one level of detail and one or more other concepts at a lower level of detail (the researchers I referenced above take the same tack). Concepts themselves are represented as predicates with associated arguments, in line with standard FOPC and theories such as case grammar [Fillmore, 1968]. One major difference in my notation is the use of a Prolog-like representation for variables which does not use explicit quantification. Bringing these threads together yields schemas such as:

$$\begin{aligned} \textit{shopping}(\textit{shopper}:S, \textit{store}:T, \textit{thing_bought}:B) \longrightarrow \\ \textit{go}(\textit{agt}:S, \textit{loc_to}:T), \textit{find}(\textit{agt}:S, \textit{obj}:B), \textit{buy}(\textit{agt}:S, \textit{obj}:B), \\ \textit{leave}(\textit{agt}:S, \textit{loc_from}:T), \textit{type}(T, \textit{shopping_place}). \end{aligned}$$

Here, the schema concept is a *shopping* event, and the slot concepts are the various events which make up a shopping event, or the *steps* of the shopping schema. The slot concepts are considered to be conjoined by the commas, which stand for \wedge . Each concept has a list of roles: for example, a *go* event has an agent (*agt*) and a location which is the destination of the event (*loc_to*).

The most difficult thing to define when writing schemas as logical expressions is the exact meaning of \longrightarrow . In logic, this is treated as an implication, specifying that ‘if the antecedent of an expression is true, then its consequent is also true’. Note that the schema is still declarative, as it doesn’t specify how information is added to a representation (as production rules do); instead it specifies a relationship which *can be employed* in inferring new information, e.g. by *modus ponens* [Copi, 1978].

Implication is quite restrictive when writing schemas for comprehension. We also want to use schemas to infer the left-hand side elements, given the right-hand side elements, on some occasions (see section 2.3.1). For this purpose, it is useful to read \longrightarrow as specifying a relationship like ‘associates with’, so that a schema can be read as ‘the antecedent and consequent elements of the schema frequently occur together’. The presence of a schema’s consequent elements in a representation can be used as evidence that the antecedent elements may also be represented (and vice versa). However, the directionality of the arrow is still important, denoting an asymmetry in the relationship: the antecedent elements provide a more precise, more specific description of the consequent elements, or a reason why they may co-occur in a representation (c.f. macrostructures [van Dijk, 1977]).

Although I started this section with the claim that logic provides a semantics and inference rules for representing schemas, I proceeded to twist this claim to admit non-logical behaviour. However, by maintaining the essentially declarative form of logical

formulae, but allowing schemas to be used in various ways by inference processes, it is possible to remove some of the procedural baggage of production rules. This emphasises the separation between the *content* of schemas and the *uses* to which they may be put.

In the next section, I describe a particular class of schemas often implicated in studies of comprehension: *scripts*. Throughout this discussion, I emphasise their status as declarative structures, using notation I have introduced in this section, with the aim of pointing out where they have been unfairly criticised by some researchers.

4.2 Scripts

Scripts have received much attention in studies of comprehension [Schank and Abelson, 1977]. Although they are now rather unfashionable, some researchers still covertly refer to them under other names, perhaps wary of appearing out-of-touch (e.g. ‘causal schemes’ in [Noordman and Vonk, 1998]).

Scripts can be thought of as schemas which possess all of the aforementioned schema characteristics, but which additionally contain event order information; this is not considered to be included in other types of schemas, such as those for categorisation [Barsalou and Sewell, 1985]. In other words, scripts contain sequencing information (normally causal) which facilitates integration of the input text into the representation [Haberlandt and Bingham, 1984], [Schank and Abelson, 1977]. Sequencing information has two effects:

- *Comprehension speed-up*: events presented in the text in same order as in the script will be comprehended more quickly [Haberlandt and Bingham, 1984]. Presumably this is because the initial activation of the script by an event A (its ‘triggering’) makes the consequent elements of the script (those ahead of A in the causal chain) more available to the processor. Subsequent events which fit these consequent elements can therefore be smoothly integrated into already-active slots, without the requirement to retrieve knowledge structures for their integration. If the event order is disrupted, as it is in [Haberlandt and Bingham, 1984], the ‘predicted’ elements do not occur and activation must be redistributed to account for the causal break.²
- *Recall improvement*: the ordering of the script’s events allows them to be recalled more readily and to be recalled in the correct order [Bower et al., 1979]. This is

²This point has never been explained to my satisfaction, so this is my own suggestion.

because a representation produced using a script contains causal information which can be utilised during recall: rather than having to recall events individually, causal chains provide a means for linking events. Once an initial event has been recalled, the other events connected to that event are also activated via these causal chains, and are thus easier to retrieve from memory.

So, the important difference between scripts and other kinds of schema is the *kind of information* that scripts contain which other types of schema do not. Other types of schema generally take the form of frames, which typically represent objects, scenes or ‘static’ entities, and lack causal ordering information [Minsky, 1975].³ However, apart from the presence of causal relations, scripts have much the same characteristics as other types of schema (see section 4.1.1). Points made about scripts can thus be taken as being applicable generally to schemas.

The reason for taking time to describe scripts separately is because they are typically associated with a certain mode of processing, generally known as *expectation-based* or *top-down* [Singer et al., 1994], [Trabasso and Magliano, 1996]. In the next section, I describe how expectation-based processing has been criticised for its failure to account for human comprehension. This failure has been taken to indicate the inviability of scripts as a theory of representation. However, I show how this criticism is based on confusion of definitions, and can be partially remedied by clarifying some key terms.

4.2.1 Are Scripts Too Top-Down?

According to Kintsch, systems based on scripts rely too heavily on top-down processing [Kintsch, 1988]. Kintsch quotes Schank’s statements about comprehension in relation to this criticism [Schank, 1978]:

We would claim that in natural language understanding, a simple rule is followed. Analysis proceeds in a top-down predictive manner. Understanding is expectation-based. It is only when the expectations are useless or wrong that bottom-up processing begins.

As Kintsch points out, expectation-based processing seems to be the exception rather than the rule. If comprehension *were* expectation-based, one would expect the comprehender to be involved in so-called ‘predict-and-substantiate’ processing [DeJong, 1979]. In

³Note that the term ‘frame’ has again been modified to describe script-like, frame-like and prototypical structures which can be dynamically modified in [Barsalou, 1992].

predict-and-substantiate processing, the comprehender typically makes many predictions about forthcoming text, which are either supported or denied by the text. However, the literature indicates that predictive inferencing is limited in scope, and highly constrained by the content of the text read so far (see [Trabasso and Magliano, 1996], [Singer et al., 1994] and section 2.3.1).

An important point is that Kintsch’s criticism applies to the *mode of processing* associated with script-based systems, rather than the actual ‘static’ structure of scripts. To use a term from computer science, the criticism is directed at the operational interpretation of scripts, rather than their declarative interpretation [Sterling and Shapiro, 1994]. To make this point clearer, consider the following rulebase:

- (1) $rob(agent:X, place:Y) \longrightarrow$
 $go(agent:X, to:Y), hold_up(agent:X, place:Y),$
 $leave(agent:X, from:Y).$
- (2) $visit(agent:X, place:Y) \longrightarrow go(agent:X, to:Y)$
- (3) $visit(agent:X, place:Y) \longrightarrow explore(agent:X, place:Y)$
- (4) $visit(agent:X, place:Y) \longrightarrow leave(agent:X, from:Y).$

If we treat these rules as declarative structures, there are several ways in which they could be applied to an input text. For example, rule (1) could be treated as a script, in the sense suggested in [Hobbs et al., 1993] (see section 4.1.2): if the event $go(agent: jack, to: wimpy)$ were observed, for instance, it would be possible to ‘trigger’ the script and infer $rob(agent: jack, place: wimpy)$. This triggering could then be used as the basis for a set of predictions: for example, forward-chaining from $rob(agent: jack, place: wimpy)$ (see section 2.3.1) would yield the two predictions $hold_up(agent: jack, place: wimpy)$ and $leave(agent: jack, from: wimpy)$. The comprehender’s aim might then be to substantiate these predictions, seeking confirmation in the remaining text. This method for applying a script treats it as a ‘cognitive unit’: activating a script activates all of the elements it contains [Walker and Yekovich, 1984].

However, it would also be possible to hold back from making any predictions, instead ignoring the unmatched consequents of the rule. Here, the script is being partially applied, and only selected elements are being activated. The inference that $rob(agent: jack, place: wimpy)$ could still be represented in memory, but the remaining component events left unspecified.

By comparison, rule (2) could be triggered to yield a single new inference, $visit(agent: jack, place: wimpy)$, based on the same $go(agent: jack, to: wimpy)$ event. This time,

there is no need to make predictions based on this rule, as its ‘slots’ have been completely filled (see section 4). However, note that ‘top-down’ predictions are possible using other rules: even though the events are not gathered into a script, there are still other possible consequences of *visit(agent: jack, place: wimpy)*, namely *explore(agent: jack, place: wimpy)* and *leave(agent: jack, from: wimpy)*.

From this simple demonstration, we can see how structures which look like scripts (rule (1)) and structures which look like links in an associative network (rules (2) - (4)) can be expressed in a similar format. The point is that scripts in themselves are not top-down, in the predictive sense; they *can* be used to perform expectation-driven comprehension, but in some senses the triggering of a script is still bottom-up. By the same token, models which employ rules representing associations between pairs of elements, like rules (2) - (4) above, are still capable of top-down inference: they can be used to produce predictions, if these kinds of inference are allowed by the algorithm. But because rules which connect pairs of elements resemble an associative network (see section 4.3) when drawn on paper, there is a tendency to forget the network’s computational equivalence to a rulebase. There is also a tendency when drawing such networks to remove the ‘arrows’ which represent the flow of information, so that there no longer appears to be a ‘top’ and ‘bottom’, only ‘associations’. Figure 4.1 demonstrates how both script-type and associative rules may be interpreted as a network. Note that the links in the diagram are non-directional, so that inferences may be made (activation passed) in either direction; if bi-directional chaining with the rules were allowed, their directionality could be overridden, yielding the same behaviour as an associative network.

So, scripts are no more ‘top-down’, in the predictive sense, than associative networks. This is the usual interpretation of top-down in the context of scripts. However, another possible meaning for the term top-down is ‘conceptually-driven’, as opposed to ‘stimulus-driven’ (bottom-up) [Eysenck and Keane, 1995]. This is a more important criticism, which I discuss in the next section.

4.2.2 Scripts and Information Accessibility

The conceptually-driven nature of scripts is derived from the way they *structure* information: they cluster concepts and make them *accessible* en masse [Walker and Yekovich, 1984]. Scripts thus support conceptually-driven comprehension by making large amounts of information available by application of a single rule. By comparison, associative rules are more data-driven, in that new information is introduced

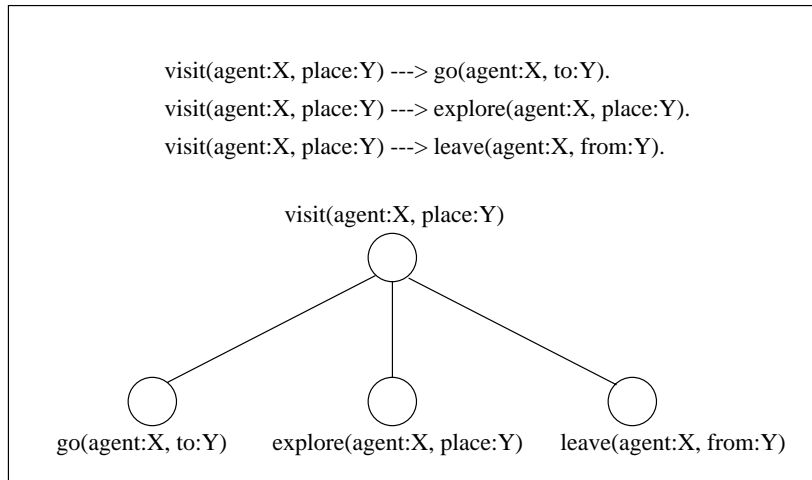


Figure 4.1: Representing rules as a semantic network

slowly and in small amounts; there is more of a tendency to stay close to the ‘data’ of the text.

The ‘size’ of a rule describes the amount of information it makes available: the greater a rule’s size, the more information it ‘makes accessible’ when it is triggered. To make this more concrete, consider some concepts related to robberies structured in two different ways: first, as a large script-like rule, relating a single higher-level element to several subordinate elements; second, as a set of smaller rules, associating pairs of elements. This yields two representations of the robbing concepts:

1. Script-like rule

$rob(agent:X, place:Y) \longrightarrow$
 $go(agent:X, to:Y), hold_up(agent:X, place:Y), leave(agent:X, from:Y).$

2. Set of associative rules

- (a) $rob(agent:X, place:Y) \longrightarrow go(agent:X, to:Y).$
- (b) $rob(agent:X, place:Y) \longrightarrow hold_up(agent:X, place:Y).$
- (c) $rob(agent:X, place:Y) \longrightarrow leave(agent:X, from:Y).$

Imagine also a comprehender who can forward- or backward-chain on rules, as defined previously, and has the capability to instantiate unmatched elements of a rule once it has been triggered (uninstantiated elements are thus *made accessible* by triggering).

For example, if a comprehender with rule (1) matches it against the event *hold_up(agent: jack, place: wimpy)*, they can infer *rob(agent:jack, place:wimpy)*; the other elements of the rule, *go(agent: jack, place: wimpy)* and *leave(agent: jack, place: wimpy)* are made accessible, and may be instantiated immediately (or left uninstantiated).

Instead, if a comprehender had rules (2a) - (2c), they could also infer *rob(agent: jack, place: wimpy)* from *hold_up(agent: jack, place: wimpy)*. However, no additional elements are made accessible by triggering the rule, as both sides of the rule have been fully instantiated. This could be remedied in subsequent inference generation cycles, by applying rules (2b) and (2c); eventually this would access all of the information clustered in rule (1).

Note that the predictiveness (in the temporal sense) of the inferences is not really the issue here. Instead, the way the rules package information determines how they make the comprehender aware of possible large-scale structures. Another way of thinking about this is in terms of how the size of a semantic representation allows the comprehender to exercise control over their comprehension. It is important to note that this doesn't necessarily mean that temporally predictive inferences *will* be made, or that inferences outside the scope of the schema *will not* be made. Rather, the clustering of information makes it possible for the comprehender to predict which information is likely to be useful for comprehension of the remaining text.

To clarify this, I discuss what I perceive as more telling differences between associative networks and schemas in the next section.

4.3 Associative Networks

As I've shown in the previous sections, the usual criticism of scripts is that they are too top-down and/or too predictive. This criticism is based on a confusion of how scripts structure information (declarative definition) and how that information may be applied (operational definition). I also showed how these issues could be separated out from one another, and how complaints about the operational definition can be countered.

However, there remains the criticism that scripts are just 'too big' and make too much information accessible per processing cycle. As this also applies to schemas in general, I will widen the discussion to encompass them.

An alternative to 'bulky' structures (schemas) in semantic memory is provided by *associative networks*. I use this term here to encompass a range of models with similar features,

such as the semantic networks of [Quillian, 1968], the event concept coherence networks of [Alterman and Bookman, 1992], and the numerous psychological models which contain networks of propositions (e.g. [Kintsch, 1988], [Trabasso and Magliano, 1996]). In terms of semantic representation, the difference between a schema and an associative network is the arrangement of the links between memory nodes:

- In a schema, links may represent *one-to-many* relationships: a single link may attach one schema-level node to several, grouped lower-level nodes.
- In an associative network, the links between nodes are *one-to-one*: each link connects only a pair of nodes.

This is the only real difference between associative networks and schemas when deciding which is the more viable description of semantic memory structures. Kintsch claims that the distinction between associative networks and schemas is that associative networks contain no ‘prestored structures’ [Kintsch, 1988]. I suggest that both contain prestored structures, but that the size of these structures is reduced in an associative network to *pairs of nodes*. The distinction is illustrated in figure 4.2.

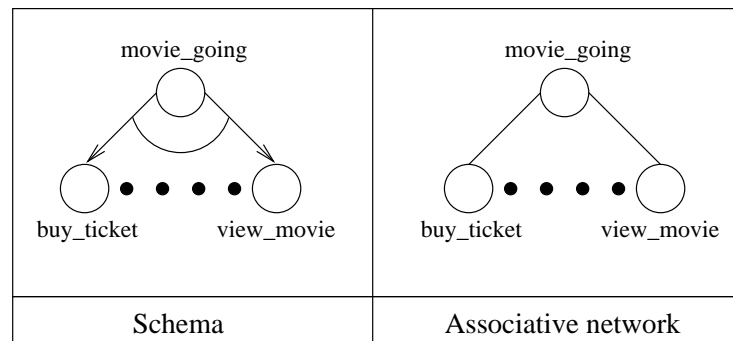


Figure 4.2: Alternative semantic representations of the same knowledge

Both (a) and (b) represent the same knowledge, in the formats introduced previously: a *movie_going* event has a *buy_ticket* event and a *view_movie* event as two of its subevents (other subevents have been omitted from the diagram) (after [Alterman and Bookman, 1990]). However, the arc in the schema illustrates that the concepts are clustered under the schema node: they are considered to be collectively implied by the presence of that node in a representation. The arrows in the schema indicate this clustering effect.

In terms of processing, this gives the benefits described in section 4.2.2: the comprehender potentially has immediate access to a bundle of related information. The *declarative* difference thus yields a processing benefit which may create *operational* differences between schemas and associative rules: clustering of information can be utilised when exercising control over comprehension, as described in the next section.

4.3.1 Control and Marker Passing

In section 4.2.1, I argued that different declarative formulations of rules could produce the same behaviour under the right kind of ‘controller’. The controller thus seems to dictate the general *operational* interpretation of the knowledge base.

However, the fact that a schema makes a cluster of information available could be exploited by the controller; in this way, the comprehender’s behaviour is influenced operationally by schema structure. For example, the controller could use the currently-active script to direct *rule retrieval* (see step 2 of figure 2.2 on page 21): that is, the currently active schema may be used to retrieve a set of potentially useful rules for processing the remainder of the text. Or, the inferences allowed may be prescribed by the content of the schema; any parts of the text which do not match slots of the schema are discarded.

By contrast, if the comprehender has rules which contain minimal structure, there is a greater tendency for irrelevant information to be inferred; often, flexibility is achieved by sacrificing control. Irrelevant information may, of course, be pruned out of a representation (for example, using the kind of integration process suggested in [Kintsch, 1988]; or by using probabilistic retrieval rules for generating associates of text propositions to limit the amount of associations made (*ibid.*)). However, in any realistic model which actually has to construct representations, there are far too many pitfalls in allowing rampant activation of associated nodes.

An example from AI illustrates this point. A particular class of comprehension models in AI is based on a theory of *marker passing* ([Alterman, 1985], [Charniak, 1983], [Norvig, 1989]). Marker passing is a computational realisation of spreading activation over an associative network; it has the following general features:

- The nodes and connections of the associative network are represented by appropriate data structures, e.g.
 - `node(shopping)`, `node(store)`.

- `connection(store_of, shopping, store)`. This connection encodes the information that the `shopping` node and the `store` node are connected by a `store_of` relationship.
- When comprehending a text, each proposition initiates the passing of a mark from the node representing the proposition to other nodes in the network.⁴ The markers thus carry activation around the network.
- Marker passing involves *copying* a marker at one node to create a new marker at another node of the associative network. Each marker has a level of activation and a record of the path from its origin node to its current location. For example, if a marker were passed between the two nodes described above, the path might be:


```
path(shopping, store_of, store)
```
- Activation is degraded as a marker moves further away from its origin, and as its age increases.
- Where two markers meet at a node, a *path* connecting the origins of the two nodes is returned.
- Inferences are made either by using the returned paths directly, or by deriving a representation from the most useful paths.

This is a generalised description of marker passing, mainly derived from [Charniak, 1986].

All marker passing models suffer from a variant of the *promiscuity* problem described in section 2.3. Often, too many paths are returned, most of which prove to be worthless (Norvig cites a figure of 10% useful paths returned [Norvig, 1989]). One particular problem occurs where concepts have a large number of links attached to them; for example, a class like ‘mammal’ is connected by *isa* relations to many animal classes, such as ‘dog’, ‘human’, and ‘cat’. Marker passing has a tendency to find many paths between concepts which are *isa* related, but this information is seldom useful: given the sentence ‘John walked his dog’ the information that both John and his dog are mammals is not particularly enlightening. However, this is the kind of inference which unbounded marker passing can return.

What is causing this problem? Returning to the idea of information clustering, marker passing suffers from a lack of *context recognition*: because the marker passing mechanism

⁴In Charniak’s formulation, markers can also be passed from individual words.

treats each node in isolation from others in the representation, there are few clues as to which groups of propositions from the text are likely to ‘belong together’. (This problem also feeds into probabilistic methods for rating interpretations, as described in section 5.1.5.)

By contrast, if schemas are used in a comprehension system, the retrieval of information as a cluster suggests the kinds of relationships which may be found during comprehension of subsequent text. In fact, scripts were initially developed as mechanism for controlling the kind of promiscuity exhibited by marker passing systems: by suggesting the kinds of information to be expected in the remaining text, inferences can be directed along productive lines [Dyer et al., 1992]. However, the early realisations of these ideas were too extreme, for the following reasons:

- Too much information was contained in a script and made inaccessible to the other scripts (i.e. information about events could not be shared between scripts). The MOP was developed as an antidote to this problem [Schank, 1982].
- Expectations about the remaining text were actually asserted into the text as temporally-predictive inferences.

As an alternative to this extreme view of scripts, consider a case where the information clustered by a schema is not necessarily asserted whenever the schema is accessed. Instead, the *amount of information clustered* could direct inference generation: the volume of information made available by the schema is used to estimate the utility of applying it. Schemas which supply a few quality-increasing inferences should be asserted, while schemas which would require many assumptive inferences (e.g. temporal predictions, inferences about the types of entities involved in the schema’s events) should not be employed.

This is comparable to a form of control suggested for marker passing models, the so-called *anti-promiscuity rules* suggested in [Norvig, 1989]. There are two forms of anti-promiscuity rule, *static* and *dynamic*:

1. *Static anti-promiscuity*

Designate as ‘promiscuous’ those nodes in the network which have more than n links attached to them, where n is some constant set by the system designer. Promiscuous nodes are barred from receiving markers.

2. *Dynamic anti-promiscuity*

Run the system on a representative sample of texts; on each run, count the number of markers on each node which received markers; next, total these counts for all runs; then designate as promiscuous those nodes in the associative network which have accumulated more than n markers, where n is again set by the designer. Promiscuous nodes cannot be used as part of certain classes of inference.

In both cases, nodes which tend to occur frequently in representations are assumed to denote less useful information; the metric used to measure promiscuity of a node is thus being used to control the system's inference generation.

The extension of this idea to schemas (with some input from other formalisms) is behind my own metric for decision making during inference generation. The main differences are:

- I take advantage of the inherent structure of the knowledge base to determine the 'utility' of individual nodes. Utility measurement depends on analysis of the clusters of knowledge associated with each node.
- The utility of making an inference using a schema depends on:
 1. The utility of its component nodes.
 2. How closely those nodes match elements of the current representation.

The utility of applying a given schema is measured in terms of the *quality* the resulting inferences will add to the representation. The full realisation of this idea is left until chapter 5.

This section has given reasons why clustering of concepts within a schema is advantageous to the comprehender. Because the knowledge in a schema comes 'pre-packaged', the comprehender has an idea of the information it potentially entails. If the commitment required to infer this information is too great, the comprehender can decide not to apply the schema. In addition, if there are two or more schemas competing to represent a text, their size may be used to settle the competition.

By contrast, when using an associative network to suggest or generate inferences the controller has no knowledge about what other concepts are likely to be retrievable once a particular node is activated; activation of subsequent nodes is not directed.

However, this may not be the whole story. Is there any way in which associative networks could produce the same kinds of effects as schemas? For example, are there psychological effects associated with schema application which can be simulated by associated networks? I turn to this issue in the next section.

4.3.2 Are Schemas Necessary?

The claim made by advocates of associative networks is that the weak method of associative retrieval, coupled with an appropriate set of weights and a relaxation algorithm, can produce the same effects as a schema-based approach. One testing ground for this idea is given by [Kintsch and Mannes, 1987]. In their model, based on the Construction-Integration framework previously discussed in section 3.2.5, they attempt to show how experimental data attributed to schemas can be accounted for by a minimally-organised associative network.

The particular data they are interested in are how people generate a list of a script's subevents. They compare this task with performance on the generation of category members. For example, when subjects are asked to generate members of categories, they typically produce 'an initial burst of typical exemplars for categories via superordinate relations during unconstrained and typicality generation' [Barsalou and Sewell, 1985]. By contrast, when asked to generate script subevents, retrieval proceeds smoothly; the number of items retrieved is linear in the amount of time spent on the task [Kintsch and Mannes, 1987]. This phenomenon has been taken as evidence of the existence of scripts: it is argued that a script contains temporal sequencing information which is made available once it is activated; this then allows smooth retrieval of the information encapsulated by the script [Barsalou and Sewell, 1985].

Kintsch and Mannes demonstrate how the linear retrieval effect can be replicated by augmenting an associative network with nodes representing the temporal relationships between pairs of event nodes. For example, they add a node representing *begin(grocery_shopping, enter)*. The network is then manipulated by spreading activation and the most highly-activated nodes are assumed to be retrieved. The resulting graph shows retrieval of script subevents to be linear in the time spent on the task (see figure 6 of [Kintsch and Mannes, 1987]).

While there is no organisation in this associative network in the sense I've suggested previously (i.e. information clusters on a large scale), there is definitely structure inherent in the network. Although there is no explicit hierarchy, the fact that some nodes repre-

senting ‘script headers’ are highly connected to many other nodes allows them to pass activation quickly and efficiently to relevant parts of the network. These groups of highly-connected nodes are functionally equivalent to a schema, in the sense that once one element of the group is activated and added to the representation (e.g. a node corresponding to a script episode) the other elements of the group are virtually guaranteed to follow suit. The weights attached to the nodes in the group are such that they quickly excite each other and inhibit nodes outside the group. This model of script-like effects is closely related to the ‘cluster of closely associated concepts’ notion of [Walker and Yekovich, 1984] (despite Kintsch’s protestations to the contrary).

In a pure-symbolic model (see section 3.2.5), the same effect occurs but at a much faster rate. Once a schema is selected, its ‘activation’ is immediately maximised, while that of other schemas is immediately dampened to zero. All of the items in the schema may then be retrieved from the activated schema one after the other in order.⁵ What may take several cycles to achieve in a symbolic-connectionist model is thus achieved in a single cycle by the pure-symbolic model; this is not all good news, though, as the gradual deactivation curve of irrelevant schemas is reduced to an immediate drop from ‘activated’ to ‘deactivated’.

It seems as though script-like retrieval effects can be reproduced without immediate, equalised activation of whole groups of nodes in semantic memory (as is hypothesised to occur in schemas). It also seems as though the graceful curves associated with human behaviour are much closer to the output of associative network models than the steep, all-or-nothing activation profiles of pure-symbolic models.⁶ This is a perennial problem in trying to get pure-symbolic models to concur with human reaction-time data [Garnham, 1996].

4.3.3 Benefits of Schemas

In this chapter, it may seem that I have tried to condemn associative networks. It is important to note that this is not my aim; instead, I am reinforcing the comments already made by those who employ these forms of representation: associative activation

⁵Walker and Yekovich criticise the assumption of equal activation across all of a script’s concepts by showing that peripheral script events receive less activation than central events [Walker and Yekovich, 1984].

⁶Other models based on symbolic-connectionist frameworks show a similar correspondence with human data (e.g. [Sharkey, 1990]).

of concepts cannot cover all forms of inference, and provides a sparse model of strategic inferences in particular (see section 3.2.5).

Some of my criticisms are based on computational criteria: for example, associative networks tend to suffer from inferential promiscuity. It is difficult to see how this could always be solved by a relaxation process which takes no account of structural information, as provided by schemas.

I am also suspicious of hand-coding of the networks which are used to prove the efficacy of the Construction-Integration model. Because the process which constructs the networks is based on intuition, much of the mess which would be returned by a computational model is removed. In some of Kintsch's examples, there seems to be no rational reason why some inferences are included and others not. This is true of one case where Kintsch discusses pronoun resolution for the following text:

The lawyer discussed the case with the judge. He said "I shall send the defendant to prison".

Kintsch wants to show how the Construction-Integration model resolves the pronoun 'he' to the judge, rather than the lawyer. To this end, he constructs the network shown in figure 4.3.

Note that there is no association from $send(lawyer, defendant, prison)$ to a corresponding $sentence(lawyer, defendant)$. Why is this the case? Kintsch states that it is because 'the process of associative elaboration generated some additional information for SEND[JUDGE, DEFENDANT, PRISON] but not for SEND[LAWYER, DEFENDANT, PRISON]' [Kintsch, 1988].

But, if a computational system generated the $sentence(judge, defendant)$ inference using a rule like

$$send(A, B, C) \longleftrightarrow sentence(A, B).$$

why would the same rule not generate $sentence(lawyer, defendant)$? The reason for the exclusion of the latter inference is because the *intuitive* rules applied by Kintsch constrain the retrieval of associated nodes according to the *types* of entities involved in a proposition (c.f. the role traditionally ascribed to scripts): $sentence(lawyer, defendant)$ is not retrieved because lawyers don't sentence people.

The computational rules which could actually automate this process are not as discriminating as Kintsch, however. They would need explicit definition of the types of entities

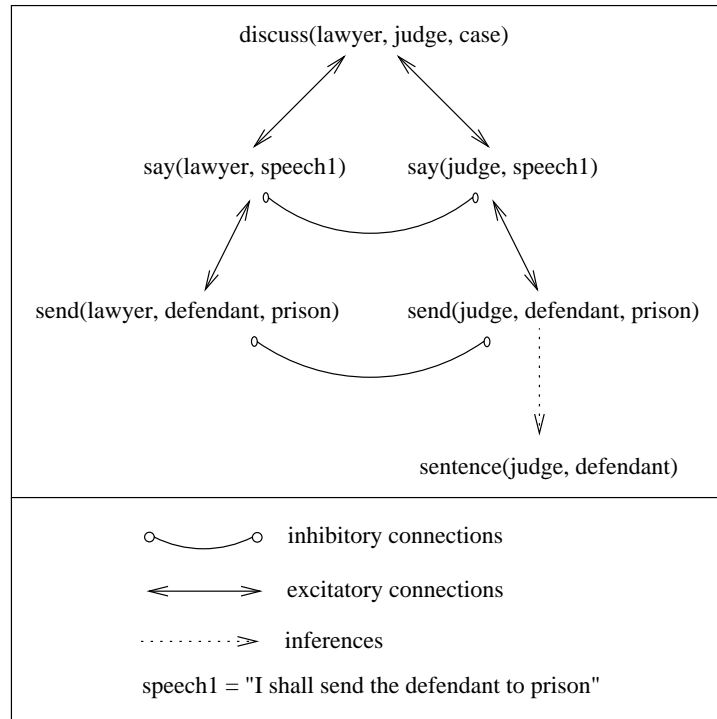


Figure 4.3: Pronoun resolution network (figure 7 of [Kintsch, 1988])

that can be involved in events. This immediately complicates the rules and increases the number required; this consequently increases the risk of inferential promiscuity.

Because I am interested in producing a computational model which both constructs and evaluates representations, I am compelled to utilise the computational benefits of schemas.⁷ This relies on using them to guide inference generation by exploiting the *amount* of information they make available to the comprehender. Because concepts are clustered into schemas the information which is common to concepts in the cluster is readily distributed: for example, if the types of several entities in a text correspond to a cluster of types defined in a schema, it is likely that the script is applicable. This commitment to schemas does not necessarily mean a commitment to excessive elaborative and predictive processing: this can be avoided by separating the operational and declarative definitions

⁷It is interesting to note that Kintsch seems to have recently admitted scripts as useful comprehension tools [Kintsch, 1998]. He even seems to be suggesting that scripts make information available in the manner I described in the section 4.3.1 (i.e. text statements processed after activation of a script can be 'slotted' into its structure).

of schemas.

4.4 Chapter Summary

A central aim of this chapter and the previous one has been definition and separation of several facets of comprehension which are often poorly defined or confused with one another, namely:

1. The reasons why interpretations are constructed.
2. The form of the episodic representations and interpretations produced.
3. The processes by which episodic representations are constructed (i.e. inferences).
4. The semantic representations used as the basis of inference generation.
5. The distinction between the operational and declarative definitions of rules in semantic memory.

Detailed descriptions of the mechanisms used to resolve these issues in my own model are given in chapter 6.

I have deliberately avoided one main issue so far which binds together several facets of comprehension: the issue of *representation quality*. In the discussion of schemas, I outlined how the amount of information contained in a schema can be used to measure the usefulness of an inference in terms of how it contributes to the quality of a representation. In the next chapter, I define how the utility of inferences can be determined by reference to their impact on representation/interpretation quality. This includes descriptions of:

- Previous attempts to define quality metrics for interpretations;
- Commonalities between those metrics;
- The role of coherence in comprehension;
- Computational models of coherence;
- A model of incoherence which attempts to unify themes from several other metrics.

Chapter 5

Metrics for Comprehension

In the previous chapters I have discussed the mechanisms underlying comprehension. An important distinction made in those chapters was between the declarative aspects of the comprehension system (e.g. static knowledge structures) and the operational aspects of the system (e.g. how those knowledge structures are applied). The framework I've developed so far depends on the following elements:

- A production system architecture based on cognitive principles.
- A knowledge base consisting of *schemas*.
- A characterisation of comprehension as reasoning with both forward- and backward-chaining.
- An abstract characterisation of *interpretations* as sets of representations which 'compress' and structure the information in a text.
- A computational description of representations as episodic networks.

Throughout I have also described the influence of control on comprehension; particularly important are the roles of quality and quality monitoring. So far, I have resisted the temptation to discuss what quality might actually be as this chapter is reserved for that purpose.

Although quality is intricately tied to the processes of many models and is thus difficult to separate out at times, I have attempted to do so in this thesis. The reasons for this are:

- Many AI systems actually have the same underlying model of comprehension processes, differing instead in their characterisation of quality: for example, abduction is the mechanism in both ACCEL [Ng and Mooney, 1992] and Wimp3 [Goldman, 1990], but the quality metrics used are coherence-based and probability-based respectively.
- In cognitive psychological models of comprehension, the actual processes by which representations are constructed vary from theoretical ([Kintsch, 1998]) to pseudo-coded ([Fletcher et al., 1996]); very few of these models have a fully-fledged inference engine. However, the metric for comparison of representations, whether implicit (see section 3.2.5) or explicit [Read and Marcus-Newhall, 1993], is often more fully developed.
- For engineering reasons, isolation of the metric in a separate ‘module’ makes program modification and testing easier. However, this can make metrics more open to criticism. When reduced to their bare bones, many of them appear quite spartan; if buried in code, they seem far more exotic and much more difficult to analyse. I have attempted to isolate previous metrics from the processes which employ them. This has a twofold effect:
 1. It allows more accurate comparisons between and description of the metrics.
 2. It allows discussion of metrics in isolation from inference processes. The tendency in papers on AI text comprehension systems is to obfuscate metrics by making them an adjunct to the description of the inference processes. Sometimes, they are even treated as an aside, almost unworthy of comment. In cognitive psychology, the tendency is more towards treatment of representations as a ‘given’ input to a connectionist relaxation metric, with little attention being given to how the structure of a representation affects how a network will relax (a notable exception being [Read and Marcus-Newhall, 1993]).

This chapter therefore attempts to discuss quality metrics with respect to some ‘given’ representations; however, I hope the previous chapters have defined sufficiently how these representations may arise.

One aim of the metric described in this chapter is to define how gross-level structural information may provide a more computationally attractive and simple model of quality than other models (such as those based on probability). In addition, incoherence also

provides an appealing account of quality monitoring which is lacking in AI models, and seems necessary given psychological data.

5.1 Representation Quality

In section 3.1, I briefly described how I view quality with respect to *economy*: the comprehender is attempting to find an interpretation for a text which relates it to as much previous knowledge as possible, allowing it to be usefully structured for recall and learning. Economy can be described in terms of possible representations, where the final interpretation compresses the text's potential entailments to within the comprehender's threshold for incoherence. The quality of the final interpretation is therefore determined by the quality of the representations of which it is composed.

This is a very general categorisation of interpretation, and one which some researchers would dispute (for example, it makes no explicit use of probability in the world). The following sections lead into a justification of my concept of *incoherence* (that is, reduction of possible structural complexity) by first describing some previous approaches to representation quality. Perhaps the most important and psychologically relevant of the approaches is based on coherence, to which I devote more time. I tentatively link each description to the later definition of incoherence, where the links become more explicit.

A brief aside: it may seem as though I have switched from talking about interpretation quality to talking about representation quality. This is because most researchers do not distinguish between an interpretation, possibly consisting of several representations, and the representation as an entity in its own right. More often, the representation and the interpretation map one-to-one onto each other; in many cases, a representation is reduced even further, to the level of an explanation of a text (in the abductive sense of section 3.2). For the purposes of this chapter, I initially talk about single representations, and assume that an interpretation is composed of a lone representation. In the next chapter, I will return to the distinction between interpretations and representations.

5.1.1 Specificity

The first characteristic of a good representation is that it should represent as specific a fact as it is possible to determine from a text [Wilensky, 1983]. Text propositions are specified by *concretion*, 'a kind of inference in which a more specific interpretation of an utterance is made than can be sustained on a strictly logical basis'. Concretion constitutes a

mechanism of ‘default classification’: vague collections of text propositions are ‘concreted’ into more substantial, integrated structures [Jacobs, 1988], [Wilensky et al., 1988].

An example due to Jacobs is based on the sentence ‘John gave a kiss to Mary’ [Jacobs, 1988]. The word ‘gave’ has a primitive semantic meaning akin to ‘transfer’; however, in this example, the more specific meaning ‘John kissed Mary’ is more ‘useful’ or *concrete*.¹ By refining the ambiguous representation ‘John transferred a kiss to Mary’ to ‘John kissed Mary’, a more accurate representation is formed.

Specificity is also an important element in the comprehension of texts. For example, representations of the following text are possible at various levels of specificity:

John went to the supermarket. He put on the uniform.

One possible representation could delineate the causal relationships between the sentences of the text: in this case, John’s travelling to the supermarket enables him to put on the uniform [van den Broek, 1990b]. Alternatively, the comprehender could infer that John went to the supermarket in order to work there [Ng and Mooney, 1990]. Another possible representation may involve the inference about John’s motivation, plus some elaborative inferences which detail John’s means of transport, what his uniform looked like, what the supermarket looked like, and so on [Whitney et al., 1991]. Each of these representations has a level of specificity associated with it, though this may be difficult to discern. Are representations describing causal interactions more or less specific than those which describe motivations? Does elaboration make a representation more specific?

A first step towards defining representation specificity may be to consider what is being specified. As I have stated before, a representation is composed both of observations and inferred propositions which explain or elaborate those observations. The latter may themselves be explained or elaborated. For example, given the text above, if a comprehender knows that John’s going to the supermarket (G) can be explained by his goal to work there (W), and his goal to work there explained by his desire to earn money (D), then the following representation could be constructed:

D explains W explains G

Depending on whether the comprehender possesses other knowledge about earning money, supermarkets, uniforms, etc., there may be other possible explanations. However,

¹I’m using the idea of ‘primitive’ semantics loosely here, in the sense suggested in [Wilensky et al., 1988].

if a proposition is an ‘end in itself’ and cannot be further explained, it could be considered *completely explained*. Similarly, if there are no other representable consequences of a proposition, it could be considered *completely elaborated*.² In the ideal (exhaustive) situation where all the possible explanations and elaborations of a text are part of its representation, the representation could be considered *maximally specific*. The specificity of a representation may therefore be defined as the level of completion of its potential entailments and elaborations: the more missing information in a representation, the greater its level of vagueness (see section 5.1.2), and the lower its specificity.

However, in practice, a comprehender doesn’t form maximally specific representations. This would require exhaustive, time-consuming inference and memory-intensive storage. Instead, representations are constructed to a particular level of specificity. Previous approaches have either ignored the possibility of varying the level of specificity or assumed that the instantiation of structures at a particular level of specificity is the aim of comprehension. By contrast, IDC is able to make more-or-less specific inferences according to how its parameters are set; the measurement and tracking of specificity is described in section 5.3.

5.1.2 Simplicity

Simplicity is important in defining good explanations: the simpler an explanation, the better the quality of that explanation. This criterion can be traced back to Occam’s Razor, which specifies criteria for deciding between explanations on the basis that ‘entities are not to be multiplied beyond necessity’ [Carroll, 1998]. In other words, the best explanations are those which require the fewest unique or new entities to be invented/assumed. This criterion is important in fields such as medical diagnosis, where an examiner will prefer to infer the fewest diseases which could cause a set of symptoms. For example, given that someone is gaining weight, has an upset stomach and is feeling tired, it may be better to infer that they are pregnant, rather than inferring that they have stopped exercising (explains ‘gaining weight’), have a stomach virus (explains ‘upset stomach’), and have mononucleosis (explains ‘feeling tired’) [Read and Marcus-Newhall, 1993]. Simplicity is also useful for assessing the quality of explanations whose *a priori* probabilities are not known (e.g. competing scientific theories; see [Thagard, 1988] and section 5.1.5).

Simplicity is a popular criterion in AI, because it can be implemented by counting the

²In a computational model, ‘completely explained’ usually means that no rule for explaining that statement is present in the knowledge base; similarly for ‘completely elaborated’.

number of assumptions required for a logical proof and comparing this with the number of elements of the input which are explained. This returns us to equation 3.1 from the last chapter, which is repeated here:

$$A \cup L \models E$$

The simplest explanation is the one which maximises

$$\frac{|E|}{|A|} \tag{5.1}$$

where A is the set of assumptions and E is the set of things explained (L denotes the set of rules ('laws') in the knowledge base). $|X|$ denotes the cardinality of the set X . Charniak uses a variant of this, where he requires that $|E| - |A|$ be greater than 0 [Charniak, 1986].

One model which uses simplicity as a central criterion is Kautz's circumscriptive theory of plan recognition [Kautz and Allen, 1986]. This model infers explanations for an agent's actions by abduction, resulting in one or more plan hypotheses; the best plan hypothesis is the one which provides a minimum covering model for the observations (i.e. an explanation requiring the minimum number of inferred plans) [Kautz, 1990].

However, simplicity alone cannot decide on the most appropriate representation in many cases. This is because the simplest representation in terms of equation 5.1 with respect to a rulebase may not be best representation according to human intuitions [Ng and Mooney, 1990]. This is because the equation doesn't consider the possibility of increasing the number of assumptions if this better ties together elements of the representation (i.e. increases coherence; see section 3.2.2).

If the number of assumptions (a measure of 'complexity') is used *in addition* to other criteria, it may allow ties produced by those criteria to be decided: given two representations which have equivalent quality by other criteria, prefer the one which requires fewest assumptions. (This is done in ACCEL; see section 5.2.4.) This shifts the emphasis from the relationship between explanation and assumption (central to equation 5.1) back to Occam's original formulation (which focuses on assumption alone).

A final point worth mentioning concerns the definition of assumption: a node is assumed if it is not explained by another node. Consider a comprehender with the following rules (after [Read and Marcus-Newhall, 1993]):

$$(1) \text{pregnant}(A) \longrightarrow \text{feeling_tired}(A), \text{gaining_weight}(A)$$

$upset_stomach(A)$.

(2) $stopped_exercising(A) \longrightarrow gaining_weight(A)$.

Given the text $\{gaining_weight(mary)\}$, there are two derivable representations, shown in figure 5.1.

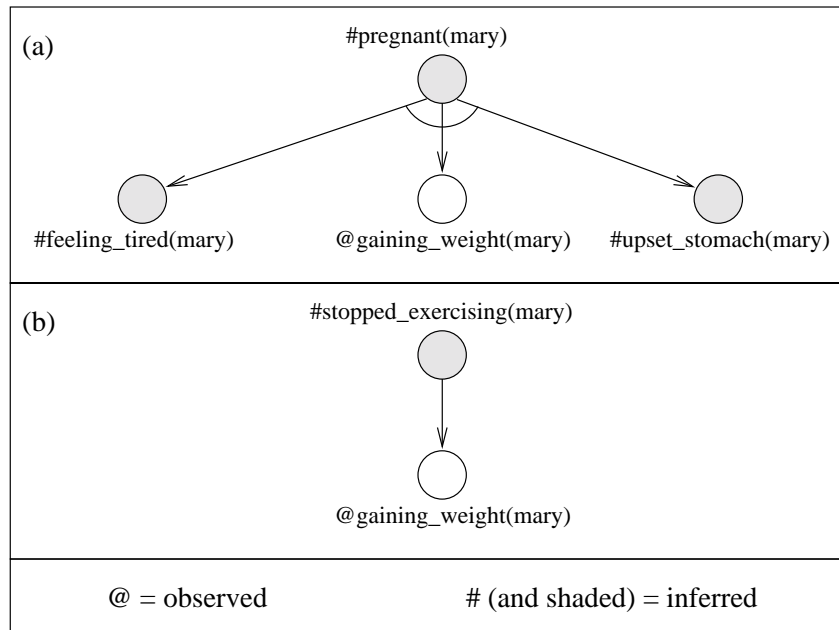


Figure 5.1: Alternative explanations for ‘gaining weight’

The simplicity criterion cannot distinguish between these representations, as both use a single assumption (each has one unexplained node); equation 5.1 assigns a simplicity rating of 1 to both. However, in representation (a), two extra inferred nodes are required to apply rule (1). The idea of assumption thus needs to be extended to cover both:

- Nodes which are not explained.
- Inferred nodes which are explained but which do not explain another node. These are more like presumptions (uncertain propositions deduced from existing propositions) than assumptions (propositions taken for granted). (The presumptions required in a representation denote the *missing information* mentioned in section 5.1.1.)

The total count of these classes of nodes then gives a measure of a representation’s reliance on uncertain nodes, or its *vagueness*. This then means that representation (a) is more vague than (b). The simplicity equation can now be rewritten as:

$$\frac{|E|}{|V|} \tag{5.2}$$

where E is the set of explained observations and $V =$ the set of assumptions and presumptions.

5.1.3 Breadth

‘This principle states that, all other things being equal, an explanatory hypothesis that explains more facts is more coherent and therefore viewed as a better explanation than an explanation that explains fewer facts.’[Read and Marcus-Newhall, 1993]

In other words, if explanation E_1 explains one observation $\{O_1\}$ and explanation E_2 explains two observations $\{O_1, O_2\}$, then E_2 has greater breadth than E_1 . Breadth may thus be calculated by counting the number of explained observations in a representation. Remember that this number was used in equation 5.1; that equation can thus now be rewritten:

$$\frac{|B|}{|V|} \tag{5.3}$$

where B (breadth) = the set of explained observations in the representation, and V the set of assumptions and presumptions. This equation calculates the overall simplicity of a representation.

Equation 5.3 makes explicit the relationship between breadth and vagueness, something which is missing from the original equation 5.1. Applying this equation gives representation (a) of figure 5.1 a simplicity of $\frac{1}{3}$, and representation (b) a simplicity of $\frac{1}{1} = 1$. This seems closer to our intuitions than the result returned by the primitive equation for simplicity (equation 5.1).

If the observation *feeling_tired(mary)* is added to the representation, representation (a)’s simplicity changes to $\frac{2}{3}$ (the inference that *mary* is pregnant explains both her feeling tired and gaining weight); while the simplicity of representation (b) changes to $\frac{1}{2}$ (as the new observation has to be assumed because there is no rule which can explain it). This is because *pregnant(mary)* is the broader explanation: it explains two observations while *stopped_exercising(mary)* explains only one. Read and Marcus-Newhall’s work has

experimentally confirmed that comprehenders prefer a single broad explanation to the conjunction of multiple narrow explanations [Read and Marcus-Newhall, 1993].

According to Thagard ([Thagard, 1988]), another area where breadth (which he terms *consilience*) is important is in the evaluation of scientific theories. In general, he claims that broad theories have a competitive edge over narrow theories. Thagard gives the example of how the general theory of relativity is more consilient than Newtonian mechanics, as the phenomena explained by the latter (the motions of planets, tides etc.) are a subset of the phenomena explained by the former (which additionally explains the bending of light by gravity, the perihelion of Mercury, etc.).

5.1.4 Competitiveness

Simplicity, derived from breadth and vagueness, is not sufficient to capture all the qualities of a good explanation. Consider the pair of representations shown in figure 5.2 (page 90).

In (a), each of a patient's symptoms is explainable by a separate hypothesis, and there is no hypothesis which can explain all three symptoms. In (b), the hypothesis that a patient is pregnant can be used to explain all three symptoms, or individual causes may still be used to explain individual symptoms. However, simplicity rates (a) and (b) with equal quality.

Read and Marcus-Newhall experimentally manipulated the number of hypotheses available to their subjects as explanations for story events ([Read and Marcus-Newhall, 1993]). They found that 'explanatory goodness' ratings for separate hypotheses were higher when no alternative, broader hypothesis was available. In other words, the goodness ratings of the separate hypotheses in (a) were found to be higher than those in (b). They attribute the discrepancy to the *competition* created by the presence of the broad hypothesis. They compare this effect to Kelley's idea of *discounting*: 'the principle that the strength of a possible cause is reduced to the extent that there are alternative plausible causes' [Kelley, 1973].

I thus hypothesise that the quality of a representation is improved to the extent that there are few or no competing representations. From where does this effect arise? In case (a), the comprehender is not aware that the *pregnant* hypothesis could explain all three symptoms (though they probably could have come to this conclusion given sufficient time); in case (b), the comprehender *is* aware of the broader explanation.

The important difference here is in what the comprehender explicitly *knows*. Computationally, this could be represented by two knowledge bases. The *pregnant* rule could be

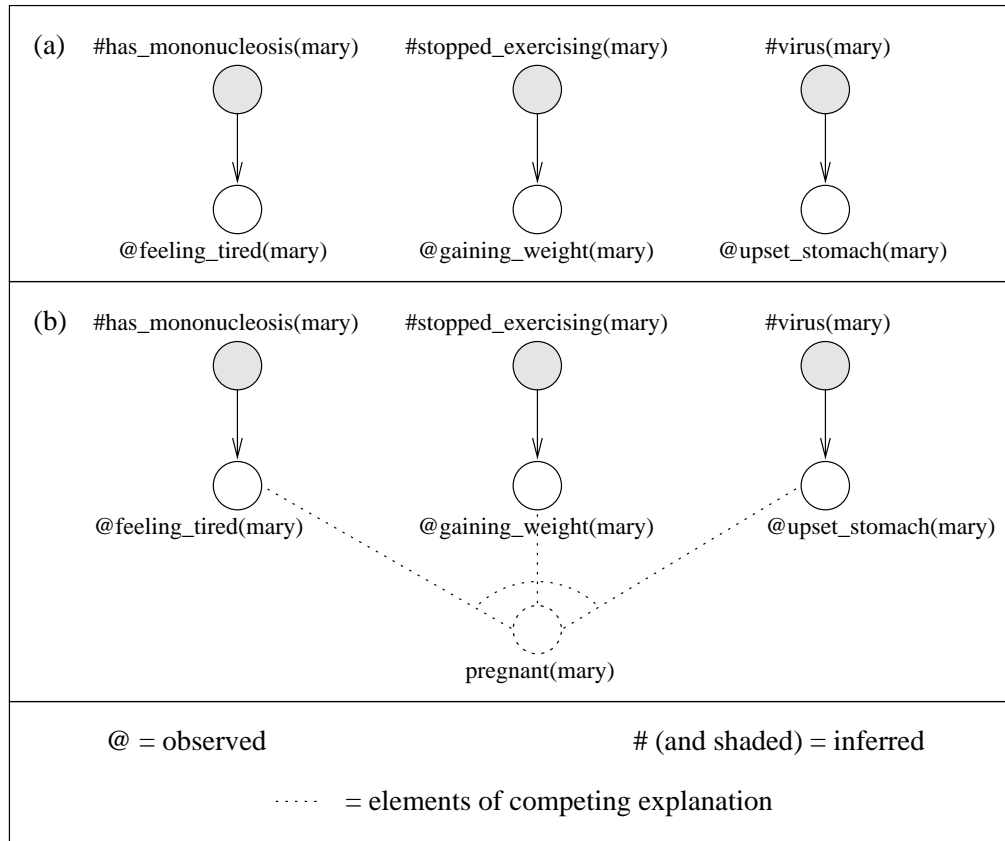


Figure 5.2: (a) Explanations without competition; (b) Explanations with competition (after [Read and Marcus-Newhall, 1993])

represented as described in section 5.1.2; this rule would be present in case (b) but absent in case (a). The system utilising the rulebase could then be given access to information about the potential explanations for each type of eventuality. In cases where there are several rules which could account for observations or previously inferred nodes, the comprehender could use this information to degrade the quality of a representation which employs one of those rules. In cases where there is only a single possible representation, quality is not degraded.

This principle is used in IDC to penalise an explanation for a set of observations if another explanation for the set of observations is possible. This is described in detail in section 5.3.5.

5.1.5 Probability

Probability theory, usually embodied in Bayesian Networks, is a popular tool for determining the quality of representations: it can be used to determine the probability that a representation is appropriate for a given text. One particular area of application within AI is plan recognition; here, the aim is ‘to choose the most likely interpretation for a set of observed actions’ [Charniak and Goldman, 1993]. The main difference between the probabilistic approach and more traditional approaches (e.g. those of section 3.2 and page 42) is that the rulebase is annotated with probabilistic information about relations between concepts. Each rule requires addition of information about the probability of its consequents, conditioned on the probability of its antecedents [Goldman, 1990]. For example, consider the following rule:

$$(1) \text{ visit}(\text{agent}:X, \text{place}:Y) \longrightarrow \\ \text{go}(\text{agent}:X, \text{to}:Y), \text{ explore}(\text{agent}:X, \text{place}:Y), \\ \text{leave}(\text{agent}:X, \text{from}:Y).$$

The rulebase is annotated with the following probabilities:

- The prior probabilities for antecedents which are not themselves the consequent of a rule. In this case, this is:

$$P(\text{visit}(\text{agent} : X, \text{place} : Y)).$$

Two values are required: the probability that $\text{visit}(\text{agent}:X, \text{place}:Y)$ is true, given no other information; and the probability that it is false. Charniak and Goldman assume that the number of events of a given type is finite with respect to the world described by a story; they then use this to estimate the prior probability that a particular type of event will occur [Charniak and Goldman, 1989]. Following their lead, reasonable estimates are:

$$\begin{aligned} P(\text{visit}(\text{agent}:X, \text{place}:Y)) \\ &= (P(\text{visit}(\text{agent}:X, \text{place}:Y)=\text{true}), P(\text{visit}(\text{agent}:X, \text{place}:Y)=\text{false})) \\ &= (10^{-7}, 1 - 10^{-7}). \end{aligned}$$

Note that the probabilities sum to 1.

- The conditional probabilities of consequents conditioned on their antecedents. In this case, these are:

$$P(\text{go}(\text{agent}:X, \text{to}:Y) \mid \text{visit}(\text{agent}:X, \text{place}:Y)) = (0.9, 10^{-6}).$$

$$P(\text{explore}(\text{agent}:X, \text{place}:Y) \mid \text{visit}(\text{agent}:X, \text{place}:Y)) = (0.8, 10^{-7}).$$

$$P(\text{leave}(\text{agent}:X, \text{from}:Y) \mid \text{visit}(\text{agent}:X, \text{place}:Y)) = (0.9, 10^{-6}).$$

The values are in the format:

$$(P(a = t \mid b = t), P(a = t \mid b = f)).$$

where t stands for ‘true’ and f for ‘false’. The remaining probabilities, $P(a = f \mid b = t)$ and $P(a = f \mid b = f)$, can be determined from the defined values:

$$P(a = f \mid b = t) = 1 - P(a = t \mid b = t).$$

$$P(a = f \mid b = f) = 1 - P(a = t \mid b = f).$$

The rulebase shown here is the most simple case. In cases where a node occurs in multiple rules, the situation becomes more complicated. The rulebase is again visualised as a network (see section 4.2.1), with each node being possibly connected to one or more parents and/or children. Each node then has an entry in the rulebase, defining its probability distribution conditional upon the probability distributions of its parents. For example, if the following rule were added to the knowledge base containing rule (1):

$$(2) \text{ rob}(\text{agent}:X, \text{place}:Y) \longrightarrow \\ \text{go}(\text{agent}:X, \text{to}:Y), \text{hold_up}(\text{agent}:X, \text{place}:Y), \\ \text{leave}(\text{agent}:X, \text{from}:Y).$$

the $\text{go}(\text{agent}:X, \text{to}:Y)$ now has two possible parents. Its probability distribution is altered to account for this and contains the following probabilities:

$$P(\text{go}(\text{agent}:X, \text{to}:Y)=t \mid \text{visit}(\text{agent}:X, \text{place}:Y)=t, \text{rob}(\text{agent}:X, \text{place}:Y)=t).$$

$$P(\text{go}(\text{agent}:X, \text{to}:Y)=t \mid \text{visit}(\text{agent}:X, \text{place}:Y)=t, \text{rob}(\text{agent}:X, \text{place}:Y)=f).$$

$$P(\text{go}(\text{agent}:X, \text{to}:Y)=t \mid \text{visit}(\text{agent}:X, \text{place}:Y)=f, \text{rob}(\text{agent}:X, \text{place}:Y)=t).$$

$$P(\text{go}(\text{agent}:X, \text{to}:Y)=t \mid \text{visit}(\text{agent}:X, \text{place}:Y)=f, \text{rob}(\text{agent}:X, \text{place}:Y)=f).$$

etc.

As can be seen from this brief example, the number of probabilities which must be defined for a node is exponential in the number of parents it has.

Other probabilities are also required:

- ‘Probability of equality’ statements, which specify the probability that two objects of the same type are actually the same object;
- Probabilities that ‘a word will be used given that the thing it denotes is of a particular type’ [Goldman, 1990], and probabilities relating to syntactic relations. I will not be discussing these, as they only apply when the input to the program is natural language.

In the next section, I describe how probability is applied to comprehension; I use Goldman’s Wimp3 program as the basis for the discussion [Goldman, 1990].

Applying Bayesian Networks to Comprehension

In systems which use probability to rate representations, comprehension is carried out in the manner described in the previous chapters: rules are applied to observations to construct representations. In this case, the representations are contained within a Bayesian network which denotes relationships between plans and the types of events they manifest. Given some observations which specify the probabilities of some of the nodes in the network, the probabilities of the other nodes can be derived using various algorithms (the one used by Goldman is Jensen’s clustering algorithm [Goldman, 1990]). For example, assuming a comprehender with rules (1) and (2) of the previous section, and the following observation:

go(agent:burt, to:l)

the network of figure 5.3 could be constructed.³ In the diagram, each node n has an associated tuple of the form $(P(n = t), P(n = f))$.

I’ve used probability values from the previous section, plus some which I left unspecified earlier. It is easy enough to distinguish the two possible plan hypotheses which could explain the observation, *rob* and *visit*. Each hypothesis may be inferred to explain the *go* observation, but because the *rob* hypothesis has a lower *a priori* probability, it is

³The probabilities in the diagram were derived using a piece of software called JavaBayes, implemented by Fabio Gagliardi Cozman and available from <http://www.usp.br/fgcozman/home.html>.

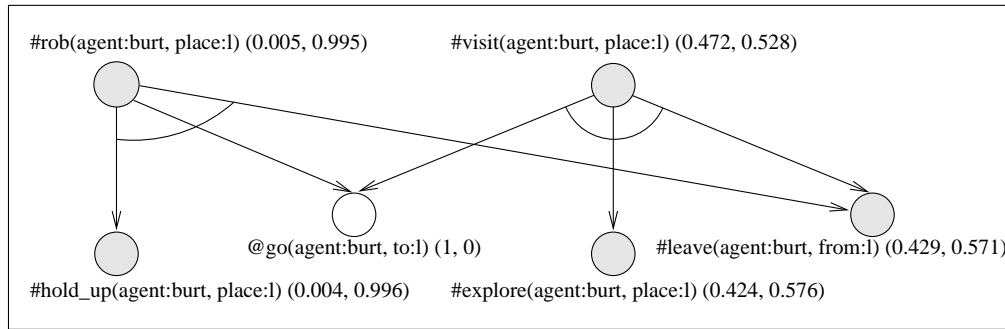


Figure 5.3: A Bayesian network representation of a text

discounted in favour of the *visit* hypothesis. However, note that multiple representations are implicit in the network, which is thus comparable to an interpretation (in the sense of section 3.2.5). Plan hypothesis nodes are privileged in Goldman’s system and the preferred representation may thus be designated as the plan hypothesis with the highest probability of being true.⁴ (Although this example focuses on plan recognition, the principle can be applied to other areas of language comprehension, such as resolution of ambiguity: see [Goldman, 1990] for examples.)

It is interesting to compare probability with other methods for measuring quality. For example, the competing explanations of figure 5.3 cannot be discriminated by equation 5.3 (page 88). Because both have the same vagueness and breadth (see sections 5.1.2 and 5.1.3), they receive the same quality rating ($= \frac{1}{3}$). By comparison, the probabilistic approach assigns a much higher probability to *visit* than *rob* by virtue of the facts that: (1) visits are more probable a priori; (2) the most important evidence for robbing (i.e. the *hold_up* event) is absent.

Note that if a *hold_up* event is observed, the approach based on breadth can make a decision between the two plan hypotheses. The probabilistic approach can also make an equivalent decision: observation of the *hold_up* node changes the probability of *rob* being true to 0.999, while the probability of *visit* being true falls to 0.0001.

Probability thus seems sensitive to both evidence which is present and evidence which is absent; it also allows the system to weight evidence according to the frequency of its

⁴It would perhaps be interesting to combine the probabilistic approach with a relaxation algorithm in the Kintsch style. The initial probabilities could then be subjected to spreading activation, so that low-probability nodes are removed from the representation entirely. As Goldman’s system stands, there is no ‘pruning’ of nodes from the representation, merely a lessening of their probability.

manifestation. While this makes the probabilistic approach seem attractive and intuitive, it rests on several assumptions with which I take issue below.

Problems with Probability as a Metric

In this section, I describe the assumptions upon which Goldman's Wimp3 program rests. This involves a fairly technical exposition of his ideas, which is necessary for my objections to become clear.

In the previous section I used various probabilities without justifying their origins. I now attempt to remedy this omission. The first question, then, is 'What do these probabilities actually mean?' The answer for priors such as $P(\textit{visit}(\textit{agent} : X, \textit{place} : Y))$ is: the probability that a given entity is of type *visit*, given a universe of discourse containing many entities of various types. This value is derived by dividing the total number of *visit* events in the universe of discourse by the total number of entities in that universe [Goldman, 1990]. A similar technique is used for all other entities, such as cars, ropes, people etc.. The general equation is therefore:

$$P(\textit{type}(X, Y)) = \frac{|Y|}{|U|} \quad (5.4)$$

Where:

- X is some entity.
- Y is a type token.
- $|Y|$ is the number of entities in the universe which are of type Y .
- $|U|$ is the total number of entities in the universe.

For example, if there are 10^{20} entities in the world and 10^9 ropes, then the probability that an entity is of type *rope* = $\frac{10^9}{10^{20}} = 10^{-11}$.

This becomes critical when trying to determine whether two entities mentioned in a discourse are of a given type. Goldman's demonstration of this is based on the (rather grisly) text:

Jack got a rope. He killed himself. [Charniak and Goldman, 1989]

The obvious inferences are that: (1) Jack killed himself by hanging; (2) the rope which Jack gets in the first sentence is the same rope with which he hangs himself. If the *get* event is designated g and the inferred *hang* event h , the probability of equality between the two ropes can be represented as:

$$P(pat_of(g) == instr_of(h) \mid type(pat_of(g), rope), type(instr_of(h), rope))$$

(i.e. the probability that the patient of (*pat_of*) the *get* event is the same entity as the instrument of (*instr_of*) the *hang* event, given that they are of the same type *rope*). Goldman sets the probability that two references to an entity of a single type actually refer to the same entity as:

$$\frac{1}{P(type(X, Y)) \times |U|} \quad (5.5)$$

where X, Y and $|U|$ are defined as previously. Also note that if equation 5.4 is substituted into this equation it becomes:

$$\frac{1}{|Y|} \quad (5.6)$$

For the case of whether two references to ropes refer to the same rope, this evaluates to $\frac{1}{10^{-11} \times 10^{20}} = 10^{-9}$. Given this evidence, the probability that a hang event *did* occur is 0.001. These are obviously very small numbers indeed; though, intuitively, a human comprehender makes this inference with little effort and considerable confidence. The problem is the system's ignorance of the fact that the events described by a text are part of a coherent discourse; as Wilensky states, 'The network treats the text as describing *completely unrelated events*' [Wilensky, 1992].

To counter this, Charniak and Goldman distinguish between the size of the set of things in the real world and the size of the set of things in the universe of discourse. They do this by making an assumption about 'spatio-temporal locality' which 'raises the probability that two things will be equal, because when restricted to a small part of space-time there are fewer different objects around' [Charniak and Goldman, 1990]. The effect of this assumption is that the number of things of a particular type shrinks; Charniak and Goldman suggest changing the number of ropes to 10, so that the probability that the two references to ropes refer to the same rope becomes $\frac{1}{10} = 0.1$ (by equation 5.6). The probability that a hang event occurred consequently rises to 0.3.

These techniques are further extended by a theory for modulating the probability that two references to an entity of a single type are references to the same entity.

Charniak and Goldman model this by setting a parameter which sets the level of likelihood for equality (i.e. in effect it changes the ‘size of the universe of discourse’) [Charniak and Goldman, 1990]. They also incorporate a theory of *mention* which increases equality probabilities: if an entity E of type T is specifically mentioned in a text, rather than inferred, then the probability that other entities of type T in the representation *are* E is increased (by some constant factor) (ibid.). My own theory of Skepticism is similar to this idea, and is discussed in section 5.3.6.

Having unravelled the complexities of Wimp3 somewhat (and I haven’t even touched upon the algorithm for handling propagation of probabilities around the network), I am now in a position to focus on the problems with probability as a quality metric.

Spatio-temporal locality is a bad assumption The spatio-temporal locality assumption seems very odd and haphazard. As Wilensky notes, it also produces strange predictions. For example, consider the following texts:

- (1) Jack went out for a meal. There was a Wimpy restaurant not far from his house.
- (2) Jack went out for a meal. There was a McDonalds restaurant not far from his house.

A fragment of a possible Wimp3 network for these texts is shown in figure 5.4.

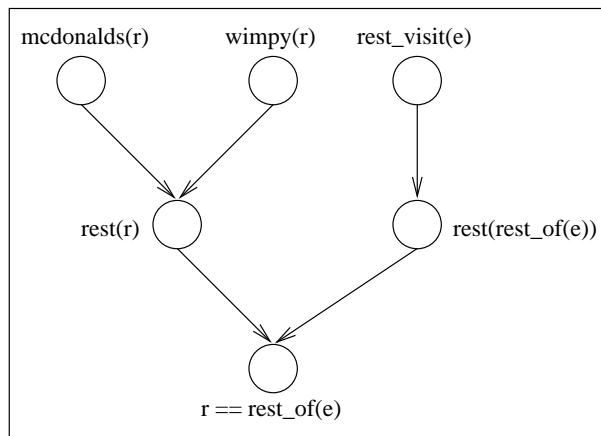


Figure 5.4: Representation of equality between restaurants

In this figure, $rest(r)$ stands for ‘ r is a restaurant’; $rest_of(e)$ is a function which returns the restaurant visited during restaurant visit ($rest_visit$) e .

As there are more McDonalds restaurants in the world than Wimpy restaurants, it seems reasonable to assume that the prior probability of the former is greater than that of the latter. However, this results in discrepancies: because the Wimpy is less probable a priori, the probability that $r == rest_of(e)$ is *higher* in text (1) than text (2). Similarly, if two instances of a type are observed in a text, they are more likely to be the same entity if there are fewer of them in the universe of discourse.

Some knowledge is not probabilistic Probability breaks down in situations where the frequency of occurrence of entities cannot be defined. For example, in assessing the comparative strengths of two scientific theories, there is no a priori way to define which is most probable [Thagard, 1989]: the scientific theories are actually determining probabilities in the world. Consider the competition between the Copernican and Ptolemaic astronomical theories: which is more probable, a priori? [Nowak and Thagard, 1992]. In the context of a world where the Ptolemaic theory (T_p) provides the paradigm, $P(T_p) = 0.99$, with competing theories each having a probability close to 0. But, once the Copernican theory T_c becomes the paradigm, the probabilities may shift so that $P(T_c) = 0.99$.

The creativity of interpretation In the case of complex narrative texts, it seems unlikely that one interpretation is more probable than another with respect to probabilities in the real world. Narrative comprehension requires a relaxation or rejection of real-world probability; more often than not, the improbable or impossible provides a narrative's topic. A more acceptable view of interpretation allies it with the construction of scientific theories (see previous paragraph). Comprehension of narratives similarly involves constructing a world where the narrative's events *become probable*. An interpretation is a 'theory of why the observable world behaves the way it does'; it is not just a theory of why the author wrote the text, but a theory of how it should be represented, how its parts interconnect, how it attaches to the historical context, and so on.

Complexity and sensitivity Hopefully, the previous paragraphs will have convinced the reader that the probabilistic approach is horrendously complicated and requires a large amount of work in defining probabilities. Some of this work may be alleviated by automation, possibly by using a technique such as Latent Semantic Analysis [Kintsch, 1998]. However, Charniak and Goldman do not discuss this possibility.

Even once probabilities have been assigned, the model is very sensitive to slight underestimation of probabilities; in particular, a change to a single probability (of the order of

0.009%) can cause massive changes in preference for one representation over another (see above and [Ng, 1992] for details).

For these reasons, I feel that probability is not sufficient as a measure of quality. It relies on defining many seemingly obscure probabilities which depend on ‘how large the universe is’. The probabilistic metric also ignores one important fact about texts: they contain entities which are related to each other, if only by virtue of being present in the same text. As a text reports on observations generated by someone (or something) other than the comprehender, relevance must be inferred: the comprehender must determine why objects and events are mentioned in the text, and provide reasons for their co-occurrence. This differs from comprehension of the real world, where pointless events may occur and irrelevant observations be made. The distinction can be summarised as:

Data in a text has been selected; data in the real world happens.

This highlights the importance of *coherence* in rating the quality of representations: the comprehender’s aim is to rationalise why the data have been chosen for inclusion in the text. The representation which defines the relatedness and connectivity of text observations must be afforded a higher quality rating than one which ignores these factors. Coherence of a representation increases as the rationale for why the data are in a text is strengthened.

This is not to say that the quality metrics considered throughout this chapter are irrelevant or wrong. Instead, I see these metrics as components which contribute to coherence. For example, the simplicity of a text depends on its vagueness, which in turn depends on the presence of unexplained nodes and inferred, explanatory nodes which are not themselves explanatory (see section 5.1.2). This can be recast as follows: if a node could be explained, but has not been, then it has *potential* explanation(s); in other words, a connection to other nodes *could* be inferred but has not yet been. A coherent representation is thus simultaneously one with a low level of vagueness, as the connections which make it coherent also realise the potential connections which would otherwise make it vague.

So far, I have discussed coherence in an intuitive fashion, by reference to connections, potential explanations, relatedness, and so on. However, this is not precise enough if one wishes to produce a computational model of coherence generation. To increase precision, the following questions must be answered:

- How is coherence judged by comprehenders and where does it originate? Is it ‘in’ the text, the representation, the comprehender, the author, or somewhere else?
- Can coherence be quantified using a technique similar to that used for simplicity, breadth etc.?

The next section tackles these questions.

5.2 Coherence

In this section, I describe several theories of coherence. Some of these theories emphasise the psychological aspects of coherence, such as the levels and quantities of coherence established during comprehension by human subjects. Others concentrate on the computation of coherence in the service of resolving competition between representations.

My emphasis is on showing how coherence is dependent on the context of comprehension. Whether a particular representation is coherent depends both on the knowledge available during comprehension, and the goals of the particular comprehension session. To this end, I am interested in abstracting away from individual structures involved in establishing coherence (e.g. causal and temporal relations), instead viewing all cognitive structures as tools for connecting r-elts. The coherence of a representation depends on which structures *could* be used to create a representation, and which structures are *actually* used.

5.2.1 Coherence in the Text

Early characterisations of coherence treated it as a property of a text. According to these theories, ‘a discourse is coherent because successive utterances are “about” the same entities’ [Hobbs, 1979]. An example of this characterisation of coherence can be found in the early work of Kintsch and van Dijk [Kintsch and van Dijk, 1978]. In their comprehension simulation, connections between successive clauses of a text are determined by argument overlap. (An argument is usually taken to mean some discourse entity such as a character or eventuality.) If two propositions derived from subsequent clauses of a text share an argument, they can be connected in the memory representation of the text. A text is coherent to the extent that it allows the formation of such links between its clauses. Consider the following:

John took a train to Paris. He arrived ten minutes late.

According to the argument overlap criterion, these two sentences are coherent because they are ‘about’ John.

However, as Hobbs points out, this characterisation of coherence does not afford sufficient explanatory power. He cites the following example in support of his claim:

John took a train from Paris to Istanbul. He likes spinach.

It is clear that this text is disjointed in some way, and would be regarded as incoherent by most readers (barring the possibility that coherence could be supported by previous text, or by ‘perverse’ reader goals). Hobbs’ alternative view is that coherence resides in the relations established between text elements by the comprehender; in other words, coherence is in the representation, rather than the text. I turn to this idea in the next section.

5.2.2 Coherence in the Representation

Instead of argument overlap, Hobbs suggests that coherence depends on the relations which obtain between pairs of text constituents, as inferred by the comprehender. The relations include *Elaboration*, *Contrast*, and *Parallel*, which are common in the literature on relations. Such relations describe semantic connections between clauses, and have been formalised by numerous writers ([Eberle, 1992], [Kehler, 1995], [Asher, 1993], [Dahlgren, 1988]). A simplified example is:

a elaborates b is true if event *a* describes event *b* in more detail. (after [Eberle, 1992])

This relation could be established between the following sentences:

John went to Germany. He was stopped at the border.
(He was stopped at the border.) *elaborates* (John went to Germany.)

Note that viewing relations as central to coherence moves the emphasis from coherence ‘in the text’, to coherence in the comprehender’s representation of the text. Relations ‘hold the representation together’.

The idea of relationships naturally extends to other forms of representational element, such as those derived from a script or other type of schema. Scripts and schemas provide a coherent, *connecting* explanation for the co-occurrence of representational elements (including elements of the text), and so contribute to the overall coherence of the representation.

Psychologists have catalogued the variety of structures which are important for establishing representational coherence. These are frequently divided into structures which form *local* coherence and those which form *global* coherence [Graesser et al., 1994]. I discuss these types of coherence in the next section.

5.2.3 Local and Global Coherence

Local coherence ‘refers to structures and processes that organize elements, constituents, and referents of adjacent clauses, or short sequences of clauses’ [Graesser et al., 1994]. To paraphrase Alterman [Alterman, 1991], it could also be called coherence-in-the-small, as it concentrates on the coherence of small amounts of low-level information.

Global coherence ‘is achieved to the extent that most or all of the constituents can be linked together by one or more overarching themes.’ [Graesser et al., 1994]. This level of coherence depends on finding ‘higher order chunks’ which capture the higher-level connections between already locally-coherent parts of the text (*ibid.*).

Much recent research has been concerned with ascertaining the amount of coherence generated in representations. This has been done by examining the kinds of information encoded into them. The classification of structures as ‘local’ or ‘global’ is then used to demonstrate the level of coherence sought by the subject: for example, Singer used a question-answering task to determine whether causal knowledge had been activated during comprehension under local coherence; he used this to demonstrate how even globally-coherent representations are constructed under conditions of local coherence [Singer and Halldorson, 1996]. This position is frequently referred to as *constructionist*. By contrast, McKoon and Ratcliff wanted to demonstrate that global coherence is not established except under conditions of local coherence breaks [McKoon and Ratcliff, 1992]. Their position is often referred to as *minimalist*.

In section 3.2.2, I made a passing comment about the functional importance of constructing certain types of representational element (r-elt): generation of associative and predictive inferences is dependent on their contribution to coherence. I take the same position as regards the local/global distinction. Rather than attempting to specify that

certain types of structure produce local coherence, and others global, I instead treat all kinds of coherence in the same manner, using the same notation etc. The main difference between r-elts is the *quantity* of coherence they generate. This depends both on the level of fit with the text (i.e. whether new r-elts have to be inferred) and the ratio of high-level elements to low-level elements.

This approach has its basis in a syntactic treatment of coherence, influenced by the work of Ng and Mooney [Ng and Mooney, 1990], [Ng and Mooney, 1992]. Their approach is detailed in the next section.

5.2.4 Quantifying Coherence in the Representation

ACCEL is an abductive plan recognition system which utilises coherence as a metric for deciding between competing interpretations [Ng, 1992]. Coherence measures ‘how well the various observations are “tied together” in the explanation’ (ibid.).

Ng’s ACCEL system directly uses the idea of coherence in its quality metric. The actual content of the representational elements is of secondary significance; what matters instead are the explanatory relationships between elements.

The metric itself is implemented by computation over directed graphs. These graphs are ACCEL’s episodic representations; the arcs in the graph represent explanations and the nodes represent propositions. The coherence of a representation is calculated by dividing the number of *actual* connections between observations (A) by the number of *possible* connections between observations (P), i.e.

$$Coherence = \frac{A}{P} \tag{5.7}$$

P is calculated as:

$$\frac{n \times (n - 1)}{2}$$

where n is the number of nodes in the graph. For any graph, this equation returns the number of possible connections between pairs of nodes, not counting connections from a node back to itself. ACCEL uses this metric to rate representations. Examples of coherence ratings are given in figure 5.5 (page 105). In both (a) and (b), the *#go – step(S, go1)* node creates a connection between the *@inst(go1, going)* and *@goer(go1, john1)* nodes (observations), so $A = 1$. As the number of nodes in each representation = 5, $P = \frac{5 \times (5 - 1)}{2} = 10$. By equation 5.7, $Coherence = \frac{A}{P} = \frac{1}{10} = 0.1$.

The coherence ratings are used to control comprehension in two ways:

1. At the end of each processing cycle, the list of current representations is pruned so that only x representations with highest coherence are maintained into the next cycle (see section 3.2.5).
2. When no more-coherent representation is produced during a cycle, that cycle is halted. Similarly, comprehension halts when the text has been fully observed and a set of most-coherent representations has been constructed.

The approach taken by Ng seems to provide a good basis for computation of quality. The metric acknowledges the importance of connectivity, yet avoids problems caused by metrics such as simplicity or competitiveness. However, ACCEL misses some aspects of coherence which I believe to be important. The central issues concern the comprehender's *context* (see section 1.3):

1. Coherence is based purely on the representation, without reference to the knowledge base from which it was constructed.
2. Coherence is determined without reference to the comprehender's quality assessment mechanism. As representations are constructed, their coherence is rated according to connectivity between observations. However, an inference's value depends *wholly* on connecting observations; if there is a way to generate at least one connection between a pair of observations, a connection is made, regardless of the existence of other potential explanations and the amount of evidence missing from the interpretation.

These problems can be demonstrated by reference to figure 5.5 (page 105) (based on rules described in [Ng, 1992]).

In (a), ACCEL has linked two observations via an inferred *shopping* trip. However, the greyed-out parts show an alternative interpretation, that a *robbing* is taking place. ACCEL prefers to infer either of the interpretations shown, rather than hold off from explaining observations. Here this seems to reduce the coherence of interpretations, as it introduces elements which have their basis in little evidence and thus seem slightly spurious. In (b), a similar situation occurs when ACCEL creates an explanation which has potential implications. Although the rules are not used deductively (as suggested by the diagram), this is where important discriminating information resides. ACCEL doesn't consider information which is possibly entailed by an explanation; if it did, it might decide that there are too many unspecified constraints to justify inference of a shopping trip (e.g. the absence of a store as a destination of the 'going' event).

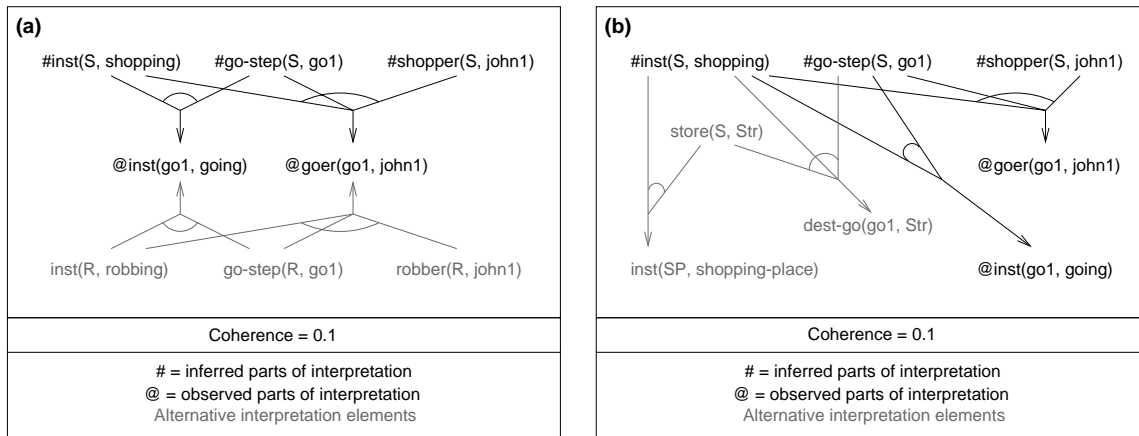


Figure 5.5: How ACCEL fails to account for potential implications

ACCEL’s quality assessment mechanism is not coupled to knowledge of the kinds of representations which are derivable. Human comprehenders do not necessarily generate every inference which could form a connection between observations (elements of a text). Instead, they are able to suppress inferences and bar them from a representation until they become appropriate [Gernsbacher et al., 1990]. The coherence of the evolving representation is not merely dependent on connections between elements, but also on ‘absent connections’ between them.

To make this more concrete, consider the following question:

Is the coherence of a representation independent of the context in which it was produced?

In figure 5.6 (on page 106), the same representation has been produced in two different contexts: in (a), the comprehender has two rules, one of which has been applied to connect observations x and y ; in (b), the comprehender has a single rule, which has also been used to connect x and y .

ACCEL assigns each of these representations the same coherence rating ($\frac{1}{3} = 0.333$); yet representation (b) is as coherent as possible, given the rulebase, while representation (a) potentially contains more structure, as another connection *could* be created. In a similar vein, (a) may contain sufficient coherence to satisfy the goals of the comprehender of (a).

My answer to the above question thus runs as follows: the coherence of a representation depends on the context in which it was generated. Deciding whether a representation

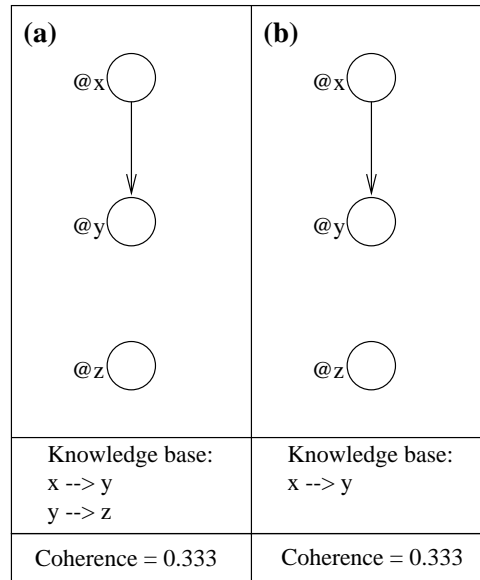


Figure 5.6: The role of the knowledge base in measuring coherence

is ‘coherent enough’ in turn depends on knowing the quantity of coherence which is potentially available; it is also reliant on an assessment mechanism which defines the proportion of potential coherence which must be established. The latter is similar to the notion of ‘coherence need’ of van den Broek et al., which they summarise as follows:

...the reader proceeds step by step through the text, at each point ascertaining whether the information that is being processed in that step has been adequately comprehended. [...] The need for coherence drives the inferential process. [van den Broek et al., 1995]

According to the ‘need for coherence’ theory, inferences are made to satisfy the comprehender’s criteria for ‘causal sufficiency’. Experimental support for the theory is given in [van den Broek et al., 1995], and several other researchers have incorporated it, or a version of it, into their own work (e.g. [Graesser et al., 1994]). These theories state that comprehension only proceeds to the point where coherence need is satisfied; this point is not necessarily that of optimum coherence. However, these researchers do not specify the terms in which this need is expressed. By viewing coherence need as the proportion of potential coherence which must be established in a representation, as I’m suggesting here, it is possible to give an abstract characterisation of the mechanisms which control comprehension.

In the next section, I describe in detail how I compute potential coherence in a representation; I do this by employing a notion of *incoherence*. I also describe how IDC decides when to make an inference based on the *structure* it adds to a representation; and how the system recognises when sufficient coherence has been established.

5.3 Incoherence

5.3.1 Basic Concepts

The basis of the incoherence metric is a fairly traditional set of knowledge sources, consisting of production-rule-like schemas with case-grammar-style roles, e.g.

- (1) `event(dine, [diner:A, place:R, thing_ordered:O, utensils:I]) →`
`[`
`event(go, [agt:A, loc_to:R]),`
`event(order, [agt:A, pat:O]),`
`event(ingest, [agt:A, pat:O, inst:I]),`
`event(pay, [agt:A, paid_for:O]),`
`type(eating_place, [exp:R])`
`].`
- (2) `event(shop, [shopper:S, store:T, thing_bought:B]) →`
`[`
`event(go, [agt:S, loc_to:T]),`
`event(find, [agt:S, pat:B]),`
`event(buy, [buyer:S, bought:B]),`
`type(shopping_place, [exp:T])`
`].`
- (3) `event(rob, [robber:A, weapon:W, place_robbed:P, thing_robbed:V,`
`victim:M]) →`
`[`
`event(get, [agt:A, pat:W]),`
`event(go, [agt:A, loc_to:P]),`
`event(point, [agt:A, pat:W, obj:M]),`
`event(get, [agt:A, pat:V, from:M]),`
`type(valuable, [exp:V]),`
`type(business, [exp:P]),`
`type(weapon, [exp:W]),`

```
habit(in_charge, [agt:M, pat:P])
].
```

The set of schemas constitutes a lattice comparable to a semantic network. The arrow in a schema is treated as a ‘connects’ relationship: the antecedent node connects the consequent nodes; or, the antecedent node is a more specific way of describing, classifying, or explaining the consequent node(s). Despite the arrows, schemas may be used for both forward- and backward-chaining. I will have more to say about the justification for and details of schemas in the next chapter (chapter 6).

I approach coherence by examining the ‘absolute incoherence’ of nodes in the schema lattice, or their ‘unrealised potential entailments’; in other words, each time an instance of a node is observed in a text, the schemas define the set of potential entailments (not in the strict logical sense) of that instance. Intuitively, incoherence corresponds to how well specified an r-elt is. For example, an r-elt representing the statement ‘Bill went to the restaurant’ is not completely specified because there are schemas available which can explain the event. Until Bill’s going to the restaurant has been explained (e.g. ‘He went because he was hungry’), the comprehender is not sure why it occurred; and until the comprehender is sure of exactly which restaurant Bill is in, how he entered it, how he got there, and so on, the ‘details’ of the representation are blurry.

In terms of the schemas above, we can *label* each node with all the higher-level nodes which can explain it. For example, the ‘Bill went to the restaurant’ r-elt can be explained by the node corresponding to ‘Bill was the agent of a dining event’; in other words, each time instances of the *event(go, [agt:bert, loc_to:r])* and *type(eating-place, [exp:r])* nodes occur in a representation, they abductively entail (or ‘can be explained by’) an instance of the *event(dine, [...])* node.⁵ Alternatively, a comprehender could elect to simply assume that those instances occurred, without inferring why. (This distinction between assumption of a node and abduction of an explanation comes from work on abductive reasoning, especially ideas from Truth Maintenance [Doyle, 1979, de Kleer, 1986].) Simultaneously, instances of all of the nodes below the *dine(...)* node also become possible; so, we can label that node with the nodes it potentially entails.

Thus, each node in the schema lattice can be thought of as having a ‘field of influence’: all of the nodes which could potentially explain it, and all of the nodes it potentially explains or ‘makes possible’. For each node *n* in the schema lattice, the size of the set of potential forward entailments (nodes ‘explained by’ *n*) and of the set of potential backward

⁵Where the details of role values are not important, I use the shorthand [...].

entailments (nodes n is ‘explained by’) can be determined. The more other nodes n explains, the higher the *informativity* of n (the more nodes n *informs* the comprehender about); the more nodes which can explain n , the higher the *ubiquity* of n (the more ubiquitous n is in representations). The chaining together of schemas also influences informativity and ubiquity: for example, if one of the forward entailments of n can itself explain the node p , then n can also indirectly explain p .

At the lowest level of the schema lattice are leaf nodes which explain no other nodes; these are the most ‘primitive’ nodes, and have a low informativity coupled with high ubiquity. They can often be explained by several higher-level events, but cannot be used to explain any lower-level events. At the highest level of the schema lattice are nodes which cannot be explained; in a sense, they require no justification and just ‘happen’. (Basic human motivations possibly fall into this class, c.f. the top-level acts of Kautz and Allen’s plan recognition system [Kautz and Allen, 1986].) These nodes have a low ubiquity, as they require no explanation; however, they have a large capacity for explanation, and thus make possible many lower-level nodes

The potential entailments of node instances in R , derived from their informativities and ubiquities, together demarcate the ‘potential representations space’ (*PRS*) of R . The size of the PRS of R is proportional to the ‘incoherence’ of R , or ‘the number of other representations which R potentially entails’. An example of the PRS of a representation is shown in figure 5.7 on page 113.

The comprehender’s aim can now be specified as follows:

Given the observations, construct an interpretation which delineates the connections between them, while at the same time introducing as few new r-elts as possible. In addition:

1. *Maintain an interpretation (I) which does not overload memory storage.*
2. *Cease making inferences when they fail to decrease the PRSs of the representations of I .*

This aim is basically a combination of the principles of coherence (connectivity) and simplicity (minimum vagueness allied with maximum explanation). A central difference is the emphasis on *control*: restriction of the size of the interpretation maintained, both in the short-term and long-term stores; and halting the inference process once a set of optimally coherent representations has been produced. Note that the latter condition is not

dependent on a global, objective criterion but on a subjective calculation of incoherence change (see section 5.3.7).

To show how these ideas work in practice, the next section gives an informal description of how IDC constructs representations, with the emphasis on incoherence. The specific mechanisms for inference processing, integrating observations etc. are detailed in chapter 6.

5.3.2 Incoherence in IDC

Comprehension in IDC proceeds as follows: statements from the text are observed and integrated into one or more evolving representations, which together constitute IDC's interpretation. For example, when comprehension begins IDC has no current representations. When the first observation is processed, a single new representation is generated, containing only that observation. This representation may then be extended by inference, to generate one or more new representations.

When the next observation is processed, it is initially integrated with each of the current representations. Then, further inferences are made, producing further representations, and so on. IDC also discards representations whose incoherence exceeds the best representations by a set amount (see section 6.4.3 on page 152).

For each representation R_i , integrating an observation introduces some new potential representations into $PRS(R_i)$. The new set of possible representations is also a PRS: this is the case because even a representation composed of a single element has a PRS. So, the new PRS of R_i can also be considered as the combined PRSs of its component elements.

As a simple example, consider the case when IDC observes the observation $O_1 = event(go, [agt:burt, loc_to:r])$. As this is the start of comprehension, IDC produces a single representation containing O_1 , which I'll call R_1 .

Next, the possible representations space of R_1 is determined by examining the representations which *could be generated from* R_1 . Using the rules of the previous section yields a PRS composed of three potential representations:

- $P_1(R_1)$: a representation where *burt* dines in the restaurant *r*.
- $P_2(R_1)$: a representation where *burt* goes shopping in store *r*.
- $P_3(R_1)$: a representation where *burt* robs the business *r*.

where $P_i(R_1)$ is the i th representation which could potentially be generated from R_1 .

Note that the potential representations here depend only on O_1 , as this is the only element in R_1 . The more potential representations introduced by an observation, the greater the incoherence of that observation.

As more observations are incorporated into R_1 , $PRS(R_1)$ increases. Once the size of $PRS(R_1)$ reaches some threshold, IDC is driven to generate inferences which reduce $PRS(R_1)$. Inferences reduce the PRS of a representation by *actualising* one or more of the potential representations and *discounting* the remaining potential representations; in other words, part of the PRS is given extra credence, while the remainder is considered less viable. In other words, IDC actualises one of the potential representations to generate R_2 , which replaces R_1 . The decision of which possible representations to actualise and which to discount depends on overlap between the possible representations introduced by elements: the best structures to actualise are those which are introduced by the most observations.

Continuing the example, imagine that the next observation $O_2 = event(order, [agt:burt, pat:b])$. This observation can be integrated into R_1 , increasing $PRS(R_1)$ by introducing the potential representation:

- $P_4(R_1)$: a representation where *burt* orders *b* in some restaurant.

The PRS of a representation is therefore considered to consist of the sum of the number of potential representations introduced by each r-elt. In this case, the only r-elts to be considered are observations, but inferred r-elts (if present) also increase the size of the PRS.

IDC is now in a position to actualise a representation and reduce the PRS of R_1 . It does this by comparing representations in the PRS. A new representation which *unifies with* the most representations in $PRS(R_1)$ is the best representation which can be derived from R_1 . In this case, a new representation R_2 which unifies with both $P_1(R_1)$ and $P_4(R_1)$ is considered the best representation to actualise; this is because there is no representation which will unify with both $P_2(R_1)$ and $P_3(R_1)$. R_2 has the form:

R_2 : *burt* dines in the restaurant *r*; he orders *b*.

The operation of actualising a representation from a PRS is shown diagrammatically in figure 5.7 on page 113. Inferred nodes are shaded and marked with ‘#’; observations are marked with ‘@’. In each of the potential representations, only the parent node of each

tree is shown; the other nodes implied by each parent are shown as a single shaded node and are considered to be present in the representation. The roles of each event either have a lower- or upper-case value: lower case values represent roles which have been instantiated by observations, while upper-case values have been created by inferences.

Each potential representation is represented as a dotted box connected to an observation by a dotted arrow. In the lower part of the diagram, a representation made potential by both observations has been accepted as a way of connecting those observations; in other words, the structure which is common to both entering the restaurant and ordering has been actualised to produce R_2 , while the other potential representations have been discounted. The resulting representation thus integrates the observations by incorporating them into a single structure; the representation also decrements the viability of the remaining potential representations, while not necessarily removing them altogether from consideration (note that $P_2(R_1)$ has now become $P_1(R_2)$, and $P_3(R_1)$ has become $P_2(R_2)$).

As I stated earlier, the incoherence of a representation is proportional to the size of the PRS; in turn, the size of the PRS is proportional to the entailments introduced by individual r-elts; thus, each r-elt can be considered to have its own incoherence. The absolute incoherence of r-elts is dependent on their informativity and ubiquity, as stated before. However, this is not the whole story, as the comprehender's *perceived incoherence* of a representation is not always equal to its absolute incoherence. The perceived incoherence of a representation R is a function of the absolute incoherence values of the elements of R (instances of nodes in the schema lattice) with respect to the comprehender's *Skepticism*. Those representations which were not actualised are considered 'discounted' in proportion to Skepticism: the higher Skepticism is, the more cautiously potential representations are discounted. Skepticism is described in greater detail in section 5.3.6.

A point worth emphasising is that incoherence is a double-edged sword. On one hand, elements of the text introduce incoherence which can be used to build new representations; they provide an impetus for the comprehension process. On the other hand, the representation built should not rest on potential structure which comes from outside the text (i.e. inferences and assumptions). As far as possible, the representation should be constructed using the structure inherent in the text.

As an analogy, imagine some workmen who are told to build a house from a pile of bricks. While it is possible to acquire more bricks and build a bigger house, it is most economical to use the bricks provided; they are a resource, as are the elements of a text.

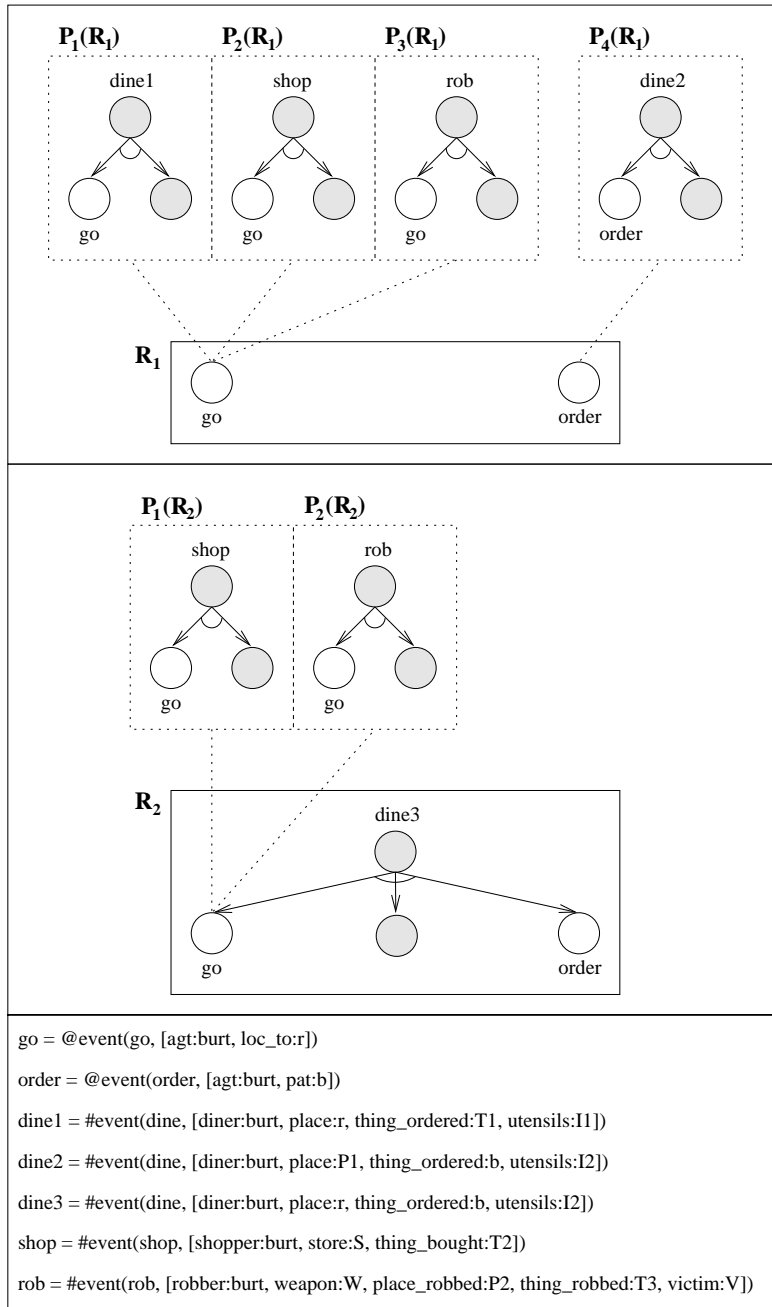


Figure 5.7: Actualising potential representations reduces the PRS

The manipulation of incoherence is thus similar to that of cognitive economy: represent the text so that it remains as distinct from representations of other texts as possible, at the same time avoiding inferences which rely on excessive assumption (c.f. [Rosch, 1978]). This seems to connect to our intuitions about text comprehension: it is not so much that we generate coherence, rather that we reduce incoherence by exploiting structure inherent in the interaction between the text and our knowledge sources.

In the next section, I explain the technical details of how informativity and ubiquity are calculated.

5.3.3 Computing Informativity and Ubiquity

In early versions of the system I used an Assumption-based Truth Maintenance System (ATMS) to perform calculation of informativity and ubiquity, modelled on ACCEL [Ng and Mooney, 1992]. At each node n , the label at n is the set of *environments* which would make n true, as defined in section 3.2.5. The length of the resulting label was then used to calculate ubiquity. In the same manner, each node can be labelled with a list of nodes which may be derived from it by deduction (forward-chaining); the length of this label is then assigned as the node's informativity.

However, it turns out that this method is unnecessarily complicated. Instead, the same measure can be derived by simply analysing how often nodes occur in the knowledge base. I compute the informativity and ubiquity of nodes in the schema lattice by noting that each node n can explain the set of nodes P and can be explained by a set of nodes Q . That is, P is the set of nodes which occur in the consequent of a schema with n as its antecedent; and Q is the set of nodes which occur as antecedents of schemas with n in their consequent.

Using this information, the following equations recursively calculate informativity and ubiquity:

$$\textit{informativity}(n) = 1 + \sum_{p \in P} \textit{informativity}(p). \quad (5.8)$$

$$\textit{ubiquity}(n) = 1 + \sum_{q \in Q} \textit{ubiquity}(q). \quad (5.9)$$

If a node cannot explain any other node, its informativity is 1; if a node cannot be explained by any other node, its ubiquity is 1.

To illustrate this, consider the simple set of schemas shown below:

$$z \longrightarrow a, b, c.$$

$$a \longrightarrow m, n.$$

These schemas can be represented diagrammatically as the schema lattice shown in figure 5.8.

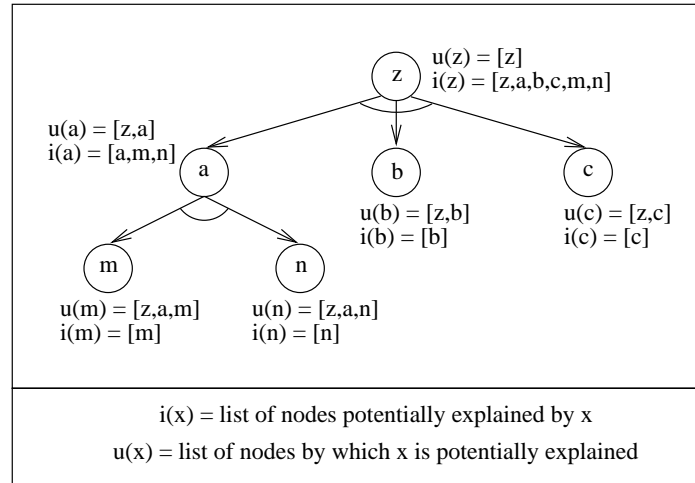


Figure 5.8: A simple schema lattice with labels

In the diagram, each node has been labelled with a *u-list* and an *i-list*. The *u-list* of node X represents those nodes which could explain X . The length of this list is used to determine ubiquity; for example, the length of $u(a)$ is 2, so the ubiquity of node a is 2. Similarly, the *i-list* at node X represents the lower-level nodes that X could explain. The length of this list is used to calculate informativity; for example, the informativity of a is 3.

These values have been determined using the ATMS method (see above), but the simpler equations 5.8 and 5.9 give the same results. The equations are applied as follows:

1. Leaf nodes of the lattice are assigned informativity = 1.
2. Root nodes are assigned ubiquity = 1.
3. Internal nodes (not leaves or roots) are assigned informativity = 1 + informativity of their immediate children, working from the leaf nodes upwards.
4. Internal nodes are assigned ubiquity = 1 + ubiquity of their immediate parents, working from the root nodes downwards.

The informativities and ubiquities of the schema nodes of section 5.3.1 are shown in table 5.1 on page 117 (as derived using the IDC setup program).⁶

In the table, the variables associated with roles have been replaced by anonymous Prolog variables. The variables are not important when determining informativity and ubiquity, but the predicate of a node and the pattern of its roles are. If a node shares a predicate and a role list with another node, then those two nodes are considered to be the same node in the schema lattice. The role lists of the two nodes must have exactly the same roles in exactly the same order for this to be true.

Having described how nodes are assigned their informativities and ubiquities, I can now describe how perceived incoherence is derived from these values. The incoherence of a representation is determined in two parts: (1) the incoherence of instances of nodes (*instances*) in the representation; (2) the incoherence of *trees* which connect instances. Both are described in the following sections.

5.3.4 Incoherence of Instances

The incoherence of instances depends on whether they *are explained by* another instance and/or whether they *explain* another instance. An instance of a node which is not explained by other instance(s) and which does not explain other instance(s) is considered more incoherent than an instance of that node which is explained and/or does explain. Incoherence also depends on whether an instance was observed or inferred: an observed instance of a node is more incoherent than an inferred instance of that node, as the presence of observed instances requires more justification than the presence of inferred instances. The latter are represented *because of* observations: they are explanations for or elaborations of observed instances, so their presence is motivated by their role in reducing incoherence. The calculation of instance incoherence (*instance_inc*) is summarised in table 5.2. Figure 5.9 on page 119 represents this information diagrammatically.

In the table, the **Explained by?/Explainer?** column denotes whether the instance is explained by another instance/explains another instance. The *S* in the formulas stands for Skepticism (see section 5.3.6 for more details). $0 < Skepticism < 1$ is always true. Note that the incoherence of observations is not affected by Skepticism. Also note that ubiquity only affects incoherence if an instance is not explained by another instance, and informativity only affects incoherence if an instance does not explain another instance.

⁶Note that these values are lower than those in the actual implementation, as other schemas employ these nodes.

Node	Informativity (inf)	Ubiquity (ubi)
event(buy, [buyer:_, bought:_])	1	2
event(dine, [diner:_, place:_, thing_ordered:_, utensils:_])	6	1
event(find, [agt:_, pat:_])	1	2
event(get, [agt:_, pat:_, from:_])	1	2
event(get, [agt:_, pat:_])	1	2
event(go, [agt:_, loc_to:_])	1	4
event(ingest, [agt:_, pat:_, inst:_])	1	2
event(order, [agt:_, pat:_])	1	2
event(pay, [agt:_, paid_for:_])	1	2
event(point, [agt:_, pat:_, obj:_])	1	2
event(rob, [robber:_, weapon:_, place_robbed:_, thing_robbed:_, victim:_])	9	1
event(shop, [shopper:_, store:_, thing_bought:_])	5	1
habit(in_charge, [agt:_, pat:_])	1	2
type(business, [exp:_])	1	2
type(eating_place, [exp:_])	1	2
type(shopping_place, [exp:_])	1	2
type(valuable, [exp:_])	1	2
type(weapon, [exp:_])	1	2

Table 5.1: Informativity and ubiquity example

Case	Type of instance I	Explained by?/ Explainer?	Incoherence
1	I is observed	no/no	$(ubi(I) - 1) + (inf(I) - 1)$
2	I is observed	no/yes	$(ubi(I) - 1)$
3	I is observed	yes/no	$(inf(I) - 1)$
4	I is observed	yes/yes	0
5	I is inferred	no/no	$(ubi(I) * S) + (inf(I) * S)$
6	I is inferred	no/yes	$(ubi(I) * S)$
7	I is inferred	yes/no	$(inf(I) * S)$
8	I is inferred	yes/yes	0

Table 5.2: Incoherence of instances

5.3.5 Incoherence of Trees

A tree is a structured object consisting of a *parent* instance and one or more *child* instances. Trees are constructed by either forward-chaining with a schema from a parent instance, or backward-chaining with a schema from a child instance. Whenever a schema is used to construct a tree, all of the elements in the tree must either be retrieved from the existing representation or inferred as new instances.

For example, given the following observation and schema:

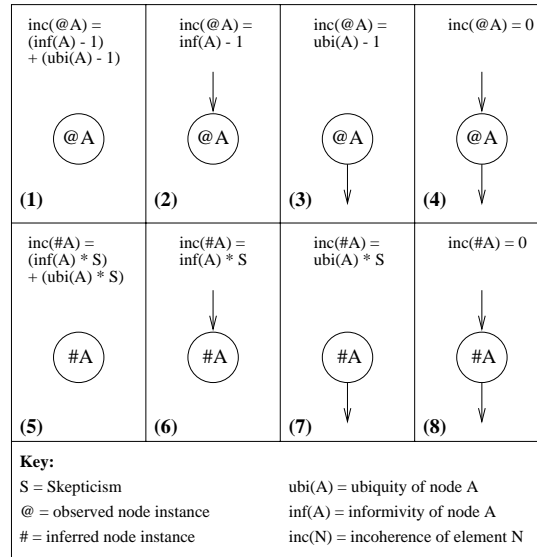
Observation:

@event(order, [agt:terry, pat:b]).

Schema:

(1) event(dine, [diner:A, place:R, thing_ordered:O, utensils:I]) \longrightarrow
 [
 event(go, [agt:A, loc_to:R]),
 event(order, [agt:A, pat:O]),
 event(ingest, [agt:A, pat:O, inst:I]),
 event(pay, [agt:A, paid_for:O]),
 type(eating_place, [exp:R])
].

this tree could be constructed:

Figure 5.9: Calculation of *instance_inc***Parent:**

#event(dine, [diner:terry, place:R, thing_ordered:b, utensils:I]).

Children:

#event(go, [agt:terry, loc_to:R]),
@event(order, [agt:terry, pat:b]),
#event(ingest, [agt:terry, pat:b, inst:I]),
#event(pay, [agt:terry, paid_for:b]),
#type(eating_place, [exp:R]).

(The exact mechanism for tree construction is left for discussion in the next chapter.)

The incoherence of a tree is measured in two parts: *altelabs_inc* and *altexpls_inc*. Both are described in the following paragraphs.

1. Incoherence caused by possible alternative elaboration sets (*altelabs_inc*)

If a tree is created which has no observed children, the tree's children have no basis in the text. The incoherence of such *groundless* trees is calculated as:

$$\text{altelabs_inc}(\text{Parent}, \text{Children}) = (\text{informativity}(\text{Parent}) - \sum_{c \in \text{Children}} \text{informativity}(c)) * S * 0.8.$$

Children is the set of inferred children of the tree; 0.8 is a constant which simulations have shown to produce the most interesting results (i.e. it ensures that trees are constructed if an instance has many possible sets of elaborations, but not if it has few sets: this prevents excessive elaborative inferencing).

If a tree does have at least one observed child, it is sufficiently justified by the text and has no incoherence.

Both situations are shown in figure 5.10.

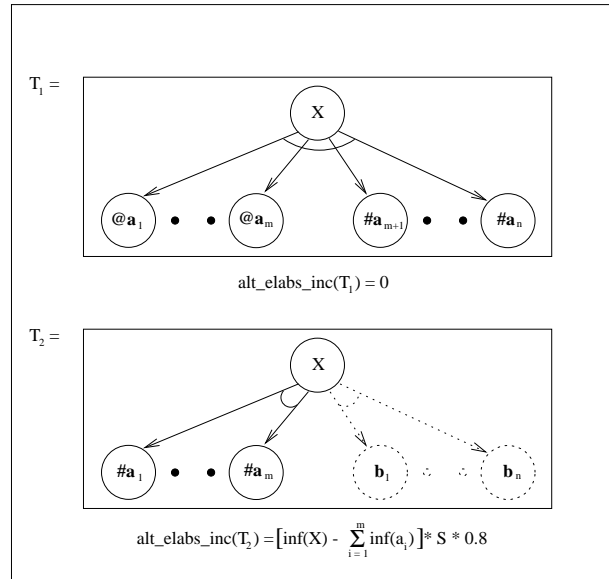
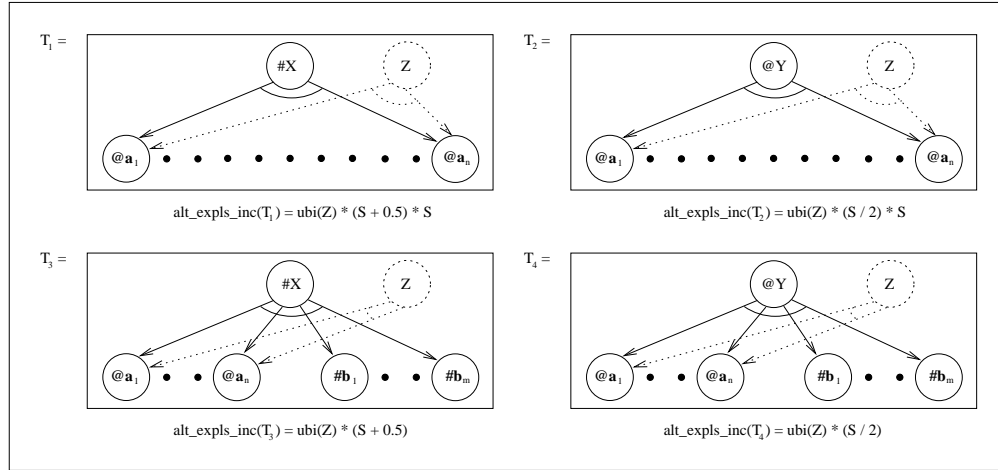


Figure 5.10: Calculation of *altelabs_inc*

2. Incoherence caused by possible alternative explanations (*altexpls_inc*) This form of incoherence is more important and complex than *altelabs_inc*. The formulae are summarised in figure 5.11 on page 121.

This figure demonstrates the four cases where *altexpls_inc* is calculated. The equations show the case where there is a single alternative explanation for the observed nodes of a tree. In the case of several parents for the tree (AltParents), *altexpls_inc* is calculated using the total ubiquity of those parents. In all the equations below, the following shorthand is used:

$$A = \sum_{a \in AltParents} ubiquity(a)$$

Figure 5.11: Calculation of $altexpls_inc$

where A is the total ubiquity of all alternative parents of the children of a tree (not including the tree's actual parent).

The details of the diagram are commented on below:

1. T_1 : a tree's parent is inferred, and all of its children are observed. In this case, if there are any other possible parents for the observed children, their ubiquity is figured into the tree's incoherence. Other possible parents are determined by looking for schemas (bar the schema used to create this tree) where all of the observed children occur in the consequent.

$$altexpls_inc(Parent, Children) = A * (S + 0.5) * S.$$

2. T_2 : a tree's parent is observed, as are all of its children. Again, other possible parents contribute incoherence according to their ubiquity, as described above; however, less incoherence is caused than in case (1), because the schema is fully instantiated by the text.

$$altexpls_inc(Parent, Children) = A * \frac{S^2}{2}.$$

3. T_3 : a tree's parent is inferred, and some of its children are inferred (i.e. they are not present in the text). This case causes the most incoherence as it rests on the

shakiest foundations: instantiation of the schema from the text is less complete than in any of the other three cases.

$$\text{altexpls_inc}(\text{Parent}, \text{Children}) = A * (S + 0.5).$$

4. T_4 : a tree's parent is observed, and some of its children are inferred. This causes less incoherence than T_3 , but more than T_1 or T_2 .

$$\text{altexpls_inc}(\text{Parent}, \text{Children}) = A * \frac{S}{2}.$$

An important point to note here is that the more information missing from a tree (i.e. the more inferred instances it contains), the greater its *altexpls_inc*. This is because trees based on inferred instances are more unstable than those based on observations. Note also that *altexpls_inc* only applies if the observed children of a tree could have been explained by some other instance: if a tree contains only inferred children, there is a penalty for *altelabs_inc* (as defined above), but no penalty for alternative explanations, as the children have been inferred for the purpose of constructing the tree. So, there is no need to consider alternative ways for explaining them.

5.3.6 Skepticism

In the previous sections, I referred to the influence of Skepticism on incoherence. Skepticism is the main means by which IDC's subjectivity is embodied in the implementation. Its role is to modulate the incoherence of r-elts produced by inference. Observations are treated as having a fixed incoherence, which is derived from the knowledge base alone; the assumption is that observations always carry a 'cognitive load' which drives the comprehender to form representations. By contrast, inferred r-elts are in a representation as explanations or elaborations of observations; Skepticism governs the extent to which these r-elts are considered spurious. The lower Skepticism is, the less cautious the comprehender is, and the more willing they are to form full representations on the basis of little evidence.

In one sense, Skepticism represents the comprehender's ability to make 'imaginative leaps' or 'jump to conclusions'. Most previous research examining this kind of behaviour has concentrated on creativity [Johnson-Laird, 1988], analogy [Forbus and Oblinger, 1990], or related areas. This work describes how existing concepts

may be combined in novel ways to produce ‘new’ ideas; often, the resulting behaviour can appear imaginative. In story comprehension research, the ISAAC system uses *creative understanding* to deal with new concepts in science fiction stories [Moorman, 1997]. Various control mechanisms decide when understanding has broken down and analogical reasoning is required to form new concepts (ibid.). However, as in previous systems which control comprehension (see section 2.4), the effort put into creating novel representations remains constant across instances of comprehension.

Skepticism is intended as a more general cognitive ability, one which directs application of schemas. Unlike ISAAC and its ilk, IDC cannot form new concepts, only apply old ones. The central difference is that IDC applies its schemas at different rates, depending on how high Skepticism is: if Skepticism is high, IDC applies schemas sparingly, reserving their use until relatively certain that it is warranted. It is thus closer to an idea like Gernsbacher’s General Comprehension Skill than it is to a ‘suite of techniques’ for guiding comprehension along novel avenues [Gernsbacher et al., 1990].

The initial impetus behind Skepticism was the idea of executive control in comprehension, influenced by [Baddeley, 1992], [Shallice, 1982], and [Cooper and Shallice, 2000]. The *central executive* is a component of working memory which ‘coordinates activity within working memory and controls the transmission of information between other parts of the cognitive system’ [Gathercole and Baddeley, 1993]. One model of the regulatory functions of the central executive is provided by Shallice’s *supervisory attentional system* (SAS) [Shallice, 1982]. During normal cognitive functioning, various competing schemas vie for control of behaviour. This routine competition is refereed by *contention scheduling*, which selects for application those schemas whose activation passes some threshold [Cooper et al., 1995]. In situations where competition cannot be resolved, the SAS may be brought in to resolve the conflict by modulating the activation of schemas, in proportion to attention focused on the schemas involved. Attending to a particular task can excite one or more schemas and inhibit their competitors.

Although the comparison between Skepticism and the SAS is perhaps slight, there are some similarities. IDC’s schemas receive ‘activation’ from the text, to the extent that they can be matched to observations: the closer the match, the better the activation of the schema, and the lower its incoherence. Skepticism increases the probability that a schema is deemed to match the text; in other words, it modulates activation of schemas and causes selection of schemas which otherwise would not pass the ‘threshold’.

In this light, a further comparison with General Comprehension Skill is pertinent.

Gernsbacher suggests that ‘Less-skilled comprehenders are less able to suppress contextually irrelevant information’ [Gernsbacher et al., 1990]. Skepticism performs a similar role, by setting standards for relevance and evidence: if Skepticism is low, irrelevant inferences are made on the basis of little evidence, which consequently have to be replaced and/or retracted. In one sense, the comprehender is unable to suppress inappropriate inferences. Conversely, high Skepticism can also be a liability: it may set such exacting standards that few inferences are made, with the result that the representation remains fragmentary.

Skepticism in Human Comprehenders

An important question is whether there is any evidence for a parameter such as Skepticism in human comprehension, and whether it manifests in empirical data. Zwaan has carried out a series of experiments which examine whether a control mechanism ‘tweaks’ comprehension, depending on the context in which it is carried out [Zwaan, 1996]. He presented the same texts to two sets of subjects, telling one set that they were news stories, and the other that they were literary stories. In the literary condition, the representations constructed, at the levels of both surface structure and textbase, were found to be richer and more elaborated. It was also found that inconsistent information was maintained more readily in the literary condition.

Zwaan infers that the differences were not caused by the texts (which were the same in both conditions), but by the comprehenders’ control systems. Using the Construction-Integration model as a basis (see pages 54 and 77), Zwaan suggests a model which is able to modulate the integration process. He describes two methods for implementing this:

1. Set a threshold at which integration is started. This threshold would be lower in literary comprehension, allowing more of the products of the construction phase to remain activated. This would also result in slower processing, as more irrelevancies and inconsistencies are maintained in the hope that they will later become relevant and be resolved.
2. Add a layer of nodes to the Construction-Integration model which is activated according to the comprehension context. For example, in a literary comprehension context, these nodes would activate those nodes in the representation which are more relevant in literature, such as surface form and textbase, while inhibiting those of lesser importance such as the situation model.

While Zwaan sides with the latter approach, my own model is closer to the former. Skepticism acts like a threshold for determining which elements remain activated and even which elements become activated in the first place, as described in the previous section. However, Skepticism only controls the ‘thoroughness’ of the inference process and not the types of products it produces (it does not discriminate between the textbase and the situation model, for example).

How Skepticism Acts as a Coherence Threshold

More evidence for a Skepticism-like parameter is present in the coherence need theory of van den Broek et al. (see section 5.2.4). It may seem that Skepticism is remote from coherence need as it is not explicitly a threshold. However, it does mimic a threshold when it is used to decide whether to produce more inferences or continue reading observations from the text. This is because IDC monitors the incoherence change caused by inferences and only accepts those which create a net decrease in incoherence. Because Skepticism influences the incoherence of inferred elements it also influences whether an inference produces a drop in incoherence or not.

Figure 5.12 gives an example of how Skepticism produces a threshold-like effect. The diagram shows a decision point where IDC has a choice between inferring an explanation (either $\#a$ or $\#b$) for event $@x$, or leaving it unexplained.

The diagram is based on the following rules:⁷

- (i) $a \longrightarrow x, y.$
- (ii) $b \longrightarrow x, z.$

In representation (1), $@x$ is not integrated into any kind of structure. This fits case 1 of table 5.2 (on page 118): an observed instance with no parents or children. Its incoherence is calculated as the instance incoherence of $@x$:

$$inc(1) = (inf(x) - 1) + (ubi(x) - 1) \quad (5.10)$$

In representations (2) and (3), x is explained by a single higher-level element (a and b respectively). Both fit case T_3 of figure 5.11; the incoherence of these representations is calculated as follows:

⁷I’ve used abstract rules to improve legibility; an example of incoherence calculation using full schemas is shown in appendix A.

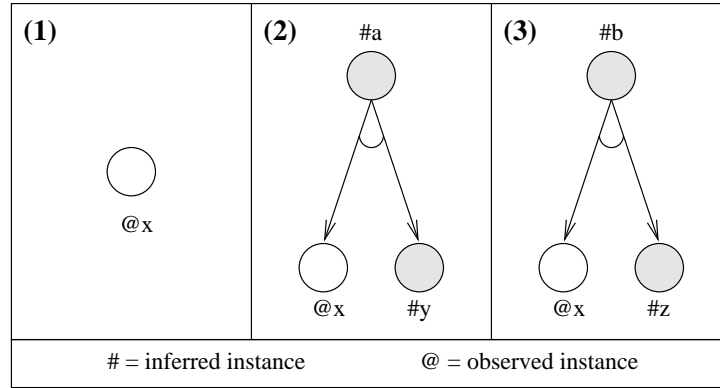


Figure 5.12: An incoherence decision point: Skepticism governs whether an explanation is inferred

$$\begin{aligned}
 inc(2) &= instance_inc(@x) + instance_inc(\#a) + instance_inc(\#y) + \\
 &\quad altexpls_inc(\#a, [@x, \#y]) \\
 &= (inf(x) - 1) + (ubi(a) * S) + (inf(y) * S) + (ubi(b) * (S + 0.5)) \quad (5.11)
 \end{aligned}$$

$$\begin{aligned}
 inc(3) &= instance_inc(@x) + instance_inc(\#b) + instance_inc(\#z) + \\
 &\quad altexpls_inc(\#b, [@x, \#z]) \\
 &= (inf(x) - 1) + (ubi(b) * S) + (inf(z) * S) + (ubi(a) * (S + 0.5)) \quad (5.12)
 \end{aligned}$$

Note that in neither case is there any *altelabs_inc*: this is because neither *a* nor *b* occurs as the antecedent of any other schema.

If rules (i) and (ii) are the only rules available to IDC, the informativities and ubiquities of the nodes in the knowledge base are as follows:

Node	Informativity (inf)	Ubiquity (ubi)
a	3	1
b	3	1
x	1	3
y	1	2
z	1	2

Using these figures, it is now possible to calculate the incoherence of each representation, as given in equations 5.10, 5.11 and 5.12.

$$inc(1) = 2 \tag{5.13}$$

$$\begin{aligned} inc(2) &= (1 - 1) + 1S + 1S + (1 * (S + 0.5)) \\ &= 3S + 1.5 \end{aligned} \tag{5.14}$$

$$\begin{aligned} inc(3) &= inc(2) \\ &= 3S + 1.5 \end{aligned} \tag{5.15}$$

IDC is thus indiscriminate between representations (2) and (3), as both require the same number of inferred nodes, at the same level of detail. However, there is a benefit in inferring representation (2) or (3) from representation (1) if such an inference produces a beneficial (i.e. negative) incoherence change. In other words, there is a benefit if:

$$\begin{aligned} inc(2) \text{ or } inc(3) &< inc(1) \\ (inc(2) \text{ or } inc(3)) - inc(1) &< 0 \end{aligned}$$

Substituting the values from equation 5.11 or 5.12 and equation 5.10 gives:

$$\begin{aligned} (3S + 0.5) - 2 &< 0 \\ 3S - 1.5 &< 0 \\ 3S &< 1.5 \\ S &< 0.5 \end{aligned}$$

When $S < 0.5$, choosing one of representations (2) or (3) is perceived as more productive than simply maintaining representation (1); when $S \geq 0.5$, representation (1) is perceived as a safer bet than either of the other two. So, Skepticism is acting as a threshold by determining when an inferred representation is perceived to be more coherent/less incoherent than a current representation.

The next section explains why incoherence change is used to determine whether a new representation should be constructed.

5.3.7 Incoherence Change

Incoherence change is used as a criterion for halting inference generation, rather than simple comparison of the coherence of representations, for one main reason: IDC's perception of incoherence is focused on the contents of its short-term store; thus, it is not measured across the whole representation (including everything in the long-term store (LTS)), but calculated only in terms of r-elts affected by an inference. The calculation may include elements in LTS, but doesn't necessarily. It is worth noting that this feature of IDC is a departure from many other coherence metrics, which consider the whole representation during evaluation (e.g. [Ng, 1992], [Thagard and Verbeurgt, 1998]). Because far fewer elements have to be evaluated when the metric is applied, the calculation is considerably faster.

Also, because incoherence in the short-term store (STS) is considered the main source of 'potential structure', it alone drives the inference process. Information which has already been processed is no longer accessible for initiation of inferences, only as a store from which elements may be 'retrieved' [Trabasso and Magliano, 1996]. However, if an inference incorporates r-elts in the LTS into a new tree, or causes LTS r-elts to become redundant, this contributes to incoherence change. For example, if a tree in the LTS is made redundant and removed, the incoherence of the r-elts unique to the tree is also removed from the representation.

As a result, it is more efficient (from a computational perspective) to measure the incoherence change of individual r-elts and sum them, rather than measure the incoherence of the representation as a whole. In the case of representations (1) - (3) above, all elements are considered equally accessible and all are affected by the inference, so changes to the LTS do not impact on incoherence change. A more complete example, demonstrating how LTS elements can impact on incoherence change, is given in appendix A; and more details of how r-elts are maintained in the STS and transferred to the LTS are given in section 6.4.1. Another factor is the number of representations maintained by IDC: for example, IDC may infer both (2) and (3) and store both in the interpretation (see section 3.2.5).

The description of incoherence has so far focused on its relationships with coherence. In the next section, I return to the other types of quality metric with which I began this chapter. In each case, I describe how the incoherence metric encapsulates parts of these metrics.

5.3.8 Specificity and Incoherence

As I stated in section 5.1.1, the specificity of a representation depends on how completely it has been explicated (i.e. explained and elaborated). In terms of the incoherence metric, any representation whose potential structure has been fully utilised (i.e. whose incoherence has been reduced to 0) is considered maximally specific. However, the only situation where a representation can be considered fully specified occurs when all of the elements of all schemas which match a text are instantiated by that text. For example, consider these two schemas (loosely based on [Ng and Mooney, 1989]):

- (i) $go_to_work(agt:A) \longrightarrow go(agt:A, loc_to:P), type(P, place_of_work),$
 $put_on(agt:A, pat:U), type(U, uniform).$
- (ii) $type(P, super_market) \longrightarrow type(P, place_of_work).$

If each of the seven nodes in these two schemas were observed in a text, both schemas could be fully instantiated and incoherence would be zero. However, it is unlikely that a text would explicitly contain all of these statements as this would become very tedious for the comprehender. Thus, normal texts leave space for the comprehender to make inferences and become engaged in representing them. Some texts leave large areas of the representation unspecified (e.g. allusive poetry [Eliot, 1958]), while others specify to a fine level of detail (e.g. the ‘nouveau roman’ [Robbe-Grillet, 1959]). However, the comprehender’s own requirement for specificity (the threshold set by their Skepticism, as described on page 125) governs how much of this ‘space’ is represented explicitly. Incoherence thus measures specificity implicitly, as incoherence is inversely proportional to specificity (more incoherent = less specific).

To show why this is the case, see figure 5.13 (page 130).

In this figure, there are three representations, (1)-(3), constructed by applying schemas (i) and (ii). In (1), neither schema has been applied; in (2), schema (i) has been applied, but schema (ii) has not; in (3), both schemas have been applied. As a rough measure, I let $specificity = A/P$, where A = the number of links between instances which have actually been specified in the representation and P = the number of links in the schema lattice.⁸ This gives the following specificities:

$$specificity(1) = 0$$

⁸No actual specificity measure has been proposed, to my knowledge, so this is an approximate measure.

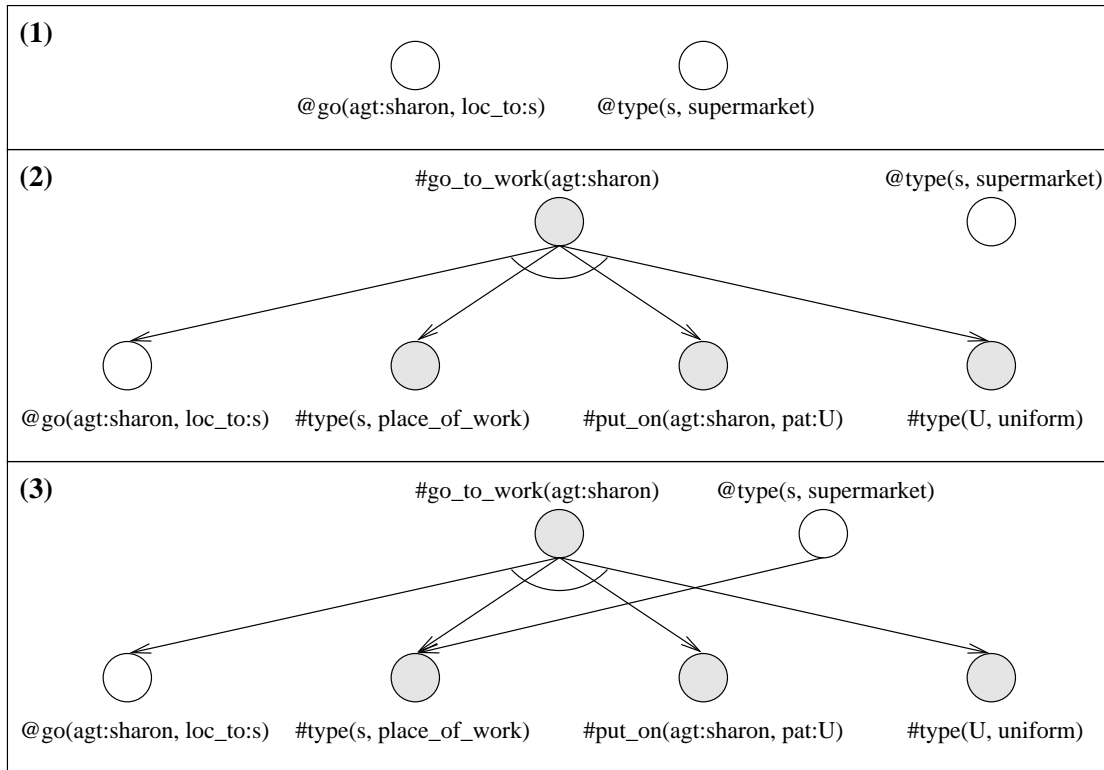


Figure 5.13: Three representations for ‘Sharon went to the supermarket’

$$specificity(2) = 4/5$$

$$specificity(3) = 1$$

In other words, representation (3) is the most specific. The incoherence metric assigns the following values to the three representations:

$$inc(1) = 2$$

$$inc(2) = 4S + 1$$

$$inc(3) = 4S$$

Representation (3) is always perceived as less incoherent than representation (2). However, unlike specificity, it is not always the case that representation (3) will be chosen over

representation (1). This is only the case when:

$$\begin{aligned} inc(3) &< inc(1) \\ 4S &< 2 \\ S &< 0.5 \end{aligned}$$

Therefore, incoherence captures some aspects of specificity, while at the same time going beyond it in recognising the importance of the comprehender's desired level of specificity (as described in section 5.1.1).

5.3.9 Simplicity, Breadth and Incoherence

Recall that in section 5.1.3 I defined simplicity in terms of *breadth* and *vagueness*: $simplicity = breadth / vagueness$. Looking again at figure 5.13, the simplicity of the three representations evaluates to:

$$\begin{aligned} simplicity(1) &= 0/2 \\ &= 0 \\ simplicity(2) &= 0.5 \\ simplicity(3) &= 0.5 \end{aligned}$$

The greater the simplicity of a representation, the better its quality, so (2) and (3) are recognised as being better representations than (1). However, simplicity cannot distinguish between (2) and (3), relying as it does on the explanation of observations for its numerator: unless more observations are explained, simplicity remains static. The incoherence metric *can* distinguish between them, as it takes account of the number of links in the representation.

A caveat: both simplicity and incoherence are reliant for their effects on the structure of the knowledge base. In systems which use simplicity, the rules are usually Horn clauses (i.e. they have multiple antecedents and a single consequent [Sterling and Shapiro, 1994]); the schemas shown above are not Horn clauses, so the metric goes slightly awry. However,

I think it is intuitively clear that representation (3) is better than representation (2), by virtue of its greater connectivity; this failing provided the impetus for Ng’s coherence metric [Ng and Mooney, 1990]. I think incoherence thus provides a better measure in these circumstances.

5.3.10 Competitiveness and Incoherence

As I’ve stated before, competitiveness is central to incoherence: the existence of competing sets of explanations and/or elaborations increases the incoherence of a representation. Referring back to figure 5.2 (page 90), the incoherence metric takes account of the difference between the knowledge bases of (a) and (b). In (a), there is no *altexpls_inc* to take into account, as there is no hypothesis which provides an alternative explanation for the observations. The incoherence of (a) = $3S$.

By contrast, in (b) there is a *pregnant(mary)* hypothesis available which can explain all three observations. Thus, there is additional *altexpls_inc* = $ubi(pregnant(-)) * (S + 0.5) * S$ (see figure 5.11 on page 121). The total incoherence is therefore calculated as:

$$\begin{aligned}
 inc(b) &= 3S + \textit{altexpls_inc}(b) \\
 &= 3S + (1 * (S + 0.5) * S) \\
 &= 3S + (S^2 + S/2) \\
 &= S^2 + 7S/2
 \end{aligned}$$

Whatever the value of S , (a) has lower incoherence than (b) (i.e. $3S < S^2 + 7S/2$ is always true for values of S such that $0 < S < 1$). It is also generally true that whenever a broad hypothesis for observations O_1, \dots, O_n is available, IDC prefers this over multiple narrow hypotheses which together explain O_1, \dots, O_n .

5.4 Chapter Summary

In this chapter, I introduced the idea of *representation quality*: a measure of how well a representation encapsulates a set of observations. Many previous metrics for quality have been suggested in the literature, including:

- Specificity
- Simplicity
- Breadth
- Competitiveness
- Probability
- Coherence

I drew comparisons between the various metrics, showing how some could be seen as components of others (e.g. breadth as part of simplicity). I also showed some drawbacks associated with the most popular metrics, probability and coherence. The following points are central to these criticisms:

1. Comprehension does not depend on probabilities in the real world, but on the probability of a representation being satisfactory in the context of a given knowledge base.
2. Comprehension control depends on acknowledging when an inference is productive. In turn, this depends on acknowledging the existence of the possible representations space. Without this, it is difficult to see how a comprehender can make a decision about when inferences are appropriate.
3. Individual comprehenders produce inferences at varying rates, and at varying points in comprehension. Without a mechanism for modulating the decision process (see previous point), there is no way to model this.

I suggested a metric based on incoherence which attempts to resolve some of these issues. The main influences behind the metric are:

1. Incorporating into a single metric as many aspects of representation quality as possible.
2. Incorporating the comprehender's knowledge base as a central determinant of quality.

3. Incorporating parameters which allow inferences to be made with more or less skepticism

Although coherence is a by-product of the management of incoherence, incoherence is a more inclusive concept. Coherence is often viewed as being distinct from other measures of representation quality (such as simplicity and competition) [Ng, 1992] or as being entirely constituted by those measures [Thagard and Verbeurgt, 1998]. Very few authors have tried to specify the importance of *structure*, abstracted away from other measures of quality. Ideally, this is what a theory of coherence should be. However, my reasons for employing incoherence were triggered by the realisation that coherence is meaningless outside the context of *potential structure*. Specification of ‘amount of structure’ is dependent on structure available.

The incoherence metric is a central part of the implementation of IDC. In the next chapter, I describe the implementation and how the metric is integrated into it.

Chapter 6

The Incoherence-Driven Comprehender

The Incoherence-Driven Comprehender (IDC) is a set of Prolog programs which together implement the comprehension model described in this thesis. IDC has two core modules:

1. The *IDC_setup module* sets up the external files used by the IDC module. It first produces indexes for the schemas in the knowledge base; then, it determines the ubiquity and informativity of nodes in the set of schemas using those indexes. Its output is a file containing the indexes and a set of annotated nodes.
 - Indexes are used to retrieve an appropriate schema from the knowledge base. A schema is retrieved by matching node instances against either the second or third argument of an index I ; the first argument of I , representing an appropriate schema ID, is then used to retrieve the schema proper. Indexes are purely for computational convenience: Prolog can retrieve indexes far more efficiently than it can match a node instance against the consequents of schemas.
 - Annotated nodes are data structures which store the node's content (see section 6.1) along with its informativity and ubiquity, determined as described in the previous chapter.
2. The *IDC module* is the comprehension model which does the main processing. It uses the indexes and annotated nodes produced by the *IDC_setup* module.

The *IDC* module is the main subject of this chapter. As a preamble to my explanation of how the module produces interpretations, I describe the types of representation it employs.

6.1 Semantic Representations: Schemas

IDC's schemas have a format familiar from the previous chapters.

```
26 : habit(care_about, [exp:A, exp:B]) / [A \== B] --->
    [
      habit(live, [exp:C, loc_in:D]) / [C ==> A],
      habit(live, [exp:E, loc_in:D]) / [E ==> B]
    ].
```

This schema represents a relationship between three nodes. The number before the antecedent (*ID*) is a unique identifier for the schema. This is used to give fast access to schemas through their indexes (see previous section).

Each node within the schema has the following form:

Eventuality_type(Predicate, Roles)

Each element of a node's content is described below.

6.1.1 Eventuality Types

The eventuality types are based very loosely on the work of Moens and Steedman (among others) [Moens and Steedman, 1988], and are limited to the following:

1. *Event*: this is used to represent eventualities which have a defined beginning and end.
2. *Habit*: this is used to represent eventualities which are indefinitely extending or 'habitual'. I used the word 'habit' rather than 'state' because 'habit' has a broader sense, encapsulating both traditional states and types. Types can be viewed as habitual states which extend over the entire lifetime of an entity, in the sense suggested in [Dalrymple, 1988].

3. *Relation*: this represents relationships between eventualities (e.g. temporal, causal, partonomic). Relations are themselves represented by instances. For example, a causal relationship between going somewhere and being at that place may be represented by the schema:

```
relation(physically_causes, [E1b, E2b]) / [] --->
[
  event(E1a, go, [agt:A, loc_to:L]) / [E1a ==> E1b],
  event(E2a, be, [agt:A, loc_at:L]) / [E2a ==> E2b],
  relation(before, [E1b, E2b]) / []
].
```

The reason for this is that it allows relations to ‘explain’ lower level eventualities.

4. *Goal*: this represents cognitive states of individuals and provides a very primitive means of describing modal statements.

Events and goals are assigned unique tokens which can be used to refer to them in relations (see section 6.1.2).

I have concentrated on events and habits as these are the easiest types of eventuality to handle. Note that I treat events at the ‘script’ level in the same way as events at other levels, in the manner of [Kautz, 1990]. Types are separated out from the role list so that information about their distribution through schemas can be used in calculating informativity and ubiquity.

I make no claims about temporal/modal reasoning, though my original intention was to incorporate both into the implementation. I quickly realised that tackling the complexities of these topics in addition to those of inferential control was beyond the scope of my thesis. As a result, the temporal ontology is very naïve and restricted.

6.1.2 Roles

Roles is a list of roles involved in the eventuality. Each role consists of a name and a value; for example, the role *agt:A* has name = *agt* and value = *A*. Role values in the schema lattice are always variables; during comprehension, these variables are instantiated where possible with values from the text, or from other inferred r-elts. The variable identifiers in a schema are important, as they influence which elements of the text can be bound to

roles, according to the schemas's constraints. The role names used in IDC are shown in table 6.1 (page 139).

In cases where an entity may be represented by two or more roles the more specific is used. For example, in 'Karen went to the garage', 'to the garage' is represented as *loc_to:garage*, rather than *pat:garage*.

In the case of relation nodes, the role list contains event tokens. For example:

```
relation(physically_causes, [E1b, E2b])
```

This role list contains tokens for two eventualities, *E1b* and *E2b*.

It is worth mentioning that the role names of table 6.1 do not limit what can be included in a schema (as the eventuality types do); rather, they merely provide a convenient shorthand. Where propositions are difficult to express using these role names, there is nothing to prevent the use of arbitrary new ones. This is especially true of script-like eventualities, which are frequently inexpressible as a solitary proposition: for example, the 'restaurant script' schema has roles such as *waiter*, *diner* and *meal*.

6.1.3 Constraints

The [Value1 Op Value2] terms attached to each node in the schema are constraints on the values taken by roles in the schema. Value1 and Value2 are two values in the schema, and Op is an operator used to compare them. Occasionally, I have also used the built-in Prolog procedure `nonvar/1` as a constraint (see below).

It is important to note that constraints are attached to schemas, not nodes; they only influence binding of variables *between* nodes within that particular schema.

The four types of constraint are:

1. `==` works as the match operator in Prolog, i.e. the constraint only holds if Value1 and Value2 are literally identical.
2. `\==` holds when Value1 and Value2 are *not* literally identical.
3. `nonvar(A)` holds if A is an instantiated variable. This constraint is usually used in conjunction with a node with eventuality type *habit*. This enforces strict conditions for application of the schema, preventing its use except in cases where an instance of the right 'habit' has been previously instantiated.

Role type	Role name	Meaning	Example
Agent	agt	The entity performing a particular action.	‘John’ - <i>agt:john</i>
Patient	pat	The entity an action is performed upon.	‘John hates Mary’ - <i>pat:mary</i>
Experiencer	exp	An entity who/which passively experiences an eventuality.	‘Tom sleeps’ - <i>exp:tom</i>
Object	obj	A subsidiary entity involved in an eventuality.	‘Frank gave the knife to Jane’ - <i>obj:knife</i>
Instrument	inst	An object which assists an entity in carrying out an action.	‘Pete ate with a fork’ - <i>inst:fork</i>
Location from	loc_from	Physical location from which a travelling action originates.	‘Paul left the house’ - <i>loc_from:house</i>
Location to	loc_to	Physical location at which a travelling action terminates.	‘Mike went to the city’ - <i>loc_to:city</i>
Location at	loc_at	Physical location where an eventuality takes place.	‘They met in the park’ - <i>loc_at:park</i>
Time of	when	Time when an event took place.	‘He arrived at noon’ - <i>when:noon</i>
Theme	thm	An event which is the subject of another event.	‘He worried about driving home’ - <i>thm:driving_home</i>

Table 6.1: Basic roles used in IDC

4. \Rightarrow is a constraint which enforces directed binding. This constraint is particularly important as it both restricts schema application and infers bindings between role values in a tree. A constraint like $[A \Rightarrow B]$ has the following effects:

- If both A and B are uninstantiated variables, they are unified with each other and the constraint holds. If one of the two variables is instantiated later, the binding thus propagates to the other variable.
- If A is an instantiated variable and B an uninstantiated variable, they are unified with each other (i.e. B takes on the same binding as A) and the constraint holds.
- If A and B are both instantiated and $A == B$ is true, the constraint holds. Note that no binding is performed here.
- If A is uninstantiated and B is instantiated, the constraint fails.

Constraints are tested in the order in which they occur in the schema as trees are built. This means that construction of a tree may be halted if a constraint fails part way through; hence, constraints limit inferences by preventing creation of trees which violate them.

6.2 Semantic Representations: Texts

The texts processed by IDC are represented in the same format as nodes; the only difference is that role values in text observations are either partially or completely instantiated. For example, the text:

John was on his way to school.

He was worried about the maths lesson.

Last week he lost control of the class. (after [Sanford and Garrod, 1981])

is represented in IDC as a list of terms:

```
event(e1, go, [agt:john, loc_to:s])
habit(school, [exp:s])
event(e2, worry_about, [agt:john, pat:m1])
habit(teacher, [exp:john])
habit(math_lesson, [exp:m1])
```

```

relation(precedes, [e1, e2])
event(e3, lost_control, [agt:john, pat:m2])
habit(math_lesson, [exp:m2])
relation(precedes, [e3, e1])
relation(precedes, [e3, e2])

```

This is necessarily a compromised representation of the text: predicates like *worry_about* and *lost_control* are not satisfactory in many respects, but dealing with the complexities of eventualities embedded within each other would distract from my central goal. As far as possible, I have tried to be consistent in my translations of texts, following the work of psychologists such as [Fletcher et al., 1996], [Kintsch, 1998], and [van den Broek et al., 1999].

Roughly speaking, each sentence in a text is translated to a single event; types involved in the sentence are specified separately using ‘habit’ eventualities; and temporal relations between events are manually derived and added to the input representation. In the latter case, it would be better if temporal relations were derived by inference from verb tenses, aspect etc.. Again, this proved to be too impractical and time-consuming.

6.3 Episodic Representations

IDC distributes each of its representations over two memory stores, the *short-term store* (STS) and the *long-term store* (LTS). These memory stores are common to most recent comprehension models [Alterman and Bookman, 1992], [Graesser et al., 1994]; their functions in IDC are described below.

6.3.1 The Short-Term Store (STS)

The STS contains the part of the representation which is currently under examination or in ‘focus’; the part ‘that the system can currently access’ [Anderson, 1983]. I have avoided use of the term ‘working memory’ as this is more often associated with a whole system of attentional mechanisms and modality-specific stores (e.g. the visuo-spatial sketch-pad, the phonological loop) [Baddeley, 1992]. Instead, I focus on the idea of a ‘conceptual buffer’ which holds information produced by activated inferences, plus recently-accessed parts of the text representation [Haberlandt and Graesser, 1990], [Jonides, 1995], [Just and Carpenter, 1992]. The STS is constantly updated during comprehension: new

r-elts are added to it from the text, new inferred r-elts are constructed in it, and r-elts are continuously retrieved into it and transferred out of it.

The capacity of the STS is determined by a *Tolerance* parameter, chiefly influenced by the work in [Just and Carpenter, 1992]. This parameter acts as a threshold against which the incoherence of STS r-elts is compared: when incoherence passes the level of *Tolerance*, the comprehender engages in reducing incoherence, either by creating inferences which utilise it or by transferring information from the STS to the LTS. *Tolerance* thus determines the amount of information the comprehender can keep in the STS from one cycle to the next.

Tolerance can be used to model different integration strategies: if *Tolerance* is low, there is a drive to integrate new information as quickly as possible into trees; if it is high, information is ‘buffered’ (maintained) until it reaches some critical mass, at which point integration is attempted [Haberlandt and Graesser, 1990]. Depending on the other parameters in the system, a low *Tolerance* may drive the comprehender to make mistakes which later have to be retracted; conversely, high *Tolerance* may disrupt the smooth flow of comprehension: because information is being maintained for long periods of time, when integration is eventually attempted there may be many possible extensions to the representation to consider.

A model which is close to this in spirit is the Current State Selection model, devised by Fletcher, Bloom and their colleagues [Fletcher et al., 1990], [Fletcher et al., 1996]. They describe the short-term memory component of the most recent version of the model as follows:

As propositions are added to short-term memory, the model focuses its attention on the propositions that are essential to the causal role played by the most recent clause that has causal antecedents, but no consequences, in the preceding text. When a sentence boundary is reached, all propositions that are not essential to this current state clause are dropped from short-term memory to make room for the following sentence. [Fletcher et al., 1996]

In a comparable fashion, IDC ‘focuses’ on those elements which are essential to the ‘structuring’ of the r-elts most recently added to the STS. It does this by centering its inferential behaviour on r-elts which have not been connected to other r-elts. R-elts which have been connected have little remaining incoherence and are purged from the STS, as they have less potential for integrating new observations. These r-elts are ‘not essential’ for integrating new observations, as their potential connections have been realised.

However, unlike the Fletcher et al. model, IDC’s ‘breadth of focus’ can be altered. If Tolerance is low, the breadth of focus is low, and information which is not immediately integrated may simply be passed into the LTS. If Tolerance is high, breadth of focus is high, and r-elts which are not relevant to the causal role of recent clauses may be maintained, simply because there is space for them in the STS. There is less of an emphasis on clausal boundaries and causality than in their model: IDC treats the potential structure of r-elts as the chief determinant of whether they should be maintained in the STS.

6.3.2 The Long-Term Store (LTS)

The LTS contains the parts of the representation which have already passed through the STS. I have not implemented sophisticated retrieval algorithms, such as those suggested in the current literature (e.g. [Myers and O’Brien, 1998]). When constructing trees, IDC simply examines the r-elts in the LTS to determine: (1) whether any of the instances there could be incorporated into the tree; (2) whether any trees have become unnecessary as a result of the new tree’s creation (see section 6.4.1). In some respects this is a primitive model of retrieval from long-term memory as it ignores ‘forgetting’ in its most general sense (e.g. there is no facilitation for recently-comprehended r-elts over those which were comprehended less recently, as is found in human comprehenders [Myers and O’Brien, 1998]).

6.3.3 Interpretations and Representations

In computational terms, the comprehender’s interpretation is a list of episodic representations. IDC thus explicitly maintains multiple representations, using the *multiple representations strategy* of figure 3.7 (page 48). The number of representations maintained depends on the Range parameter, as described in section 6.4.2. (N.B. it is also possible to set the number of retained representations to 1.)

Each representation is a data structure of the following form:

```
repr(STS, LTS, Incoherence)
```

STS and LTS each consist of a set of instances and a forest (a set of trees), i.e.:

```
STS = STS_Instances ^ STS_Forest
```

```
LTS = LTS_Instances ^ LTS_Forest
```

The \sim symbol is used to connect the instances to the forest so that they can be manipulated together in the program.

The term *r-elts* is used to collectively refer to elements of the representation, which may be either instances or trees. Instances and forests (sets of trees) are described below.

Instances

Instances are instantiated or partially-instantiated copies of nodes in the schema lattice. Two types of instance are used:

1. *Observed instances* are marked with an @ symbol. They are initially derived from the text, but may be modified later if one or more of their roles have anonymous values. For example, the observation ‘Jack was worried’ may be represented by the initial observed instance:

$$\text{instance}(@\text{event}(e2, \text{worry_about}, [\text{agt:john}, \text{pat:-}]), n, n, 10)$$

Note that the patient role has an anonymous value (-). The ‘n’ flags denote whether an instance has parents or children, in that order. The ‘10’ represents the incoherence of the instance. If a later observation specifies that Jack’s marriage is in trouble, the value of the *pat* role may be instantiated:

$$\begin{aligned} &\text{instance}(@\text{event}(e2, \text{worry_about}, [\text{agt:john}, \text{pat:m}]), n, n, 10) \\ &\text{instance}(@\text{habit}(\text{marriage}, [\text{exp:m}]) \end{aligned}$$

2. *Inferred instances* are marked with a # symbol. These are always derived by inference of trees. For example, if the comprehender inferred ‘Jack was worried’, this might be represented as:

$$\text{instance}(\#\text{event}(-, \text{worry_about}, [\text{agt:john}, \text{pat:-}]), y, n, 5)$$

If a later observation can be merged with an inferred instance, the symbol for the instance is updated, as is its incoherence. Merging of a new observation with an existing inferred instance uses the \Rightarrow constraint, specified in section 6.1.3. This ensures that bindings gravitate from observations to inferred instances: to merge observation *O* with inferred instance *I*, the following constraints must hold:

- (a) If O and I are both of type ‘event’ or both of type ‘goal’, and the eventuality token of $O = t_O$ and that of $I = t_I$, $t_O ==> t_I$ must be true. In addition, their predicates and roles must unify (using Prolog’s ‘=’ operator).
- (b) If both O and I are of type ‘relation’, their predicates must unify and their role lists match (i.e. the role list of $O ==$ the role list of I).
- (c) If both O and I are of type ‘habit’, their predicates and role lists must unify.

These constraints are necessary to prevent tokens being bound to the incorrect events and goals. There is a good possibility that complex texts would require more complex constraints, but these are sufficient for my purposes.

The procedure can be demonstrated by showing how an inferred instance may be merged with a new observation:

Inferred instance:

instance(#event(–, worry_about, [agt:john, pat:–]), y, n, 5)

Observation:

event(e3, worry_about, [agt:john, pat:m])

New instance:

instance(@event(e3, worry_about, [agt:john, pat:m]), y, n, 3)

Forests

A forest is a set of trees describing relationships between instances. An important point is that instances in a forest in one memory store are not necessarily stored in that memory store: for example, a tree in the STS forest may contain a reference to an instance which currently resides in the LTS.

As described previously, a tree is a structured object representing the relationship between a *parent* node and the *child* instances it subsumes (see section 5.3.5).

6.4 Comprehension Processes

The main comprehension process is described in this section. At the top level, this procedure makes gross-level decisions about what action to take next: whether to halt comprehension, read in the next observation, or process the current representations in an

effort to lower incoherence. This decision is made at the beginning of each *comprehension cycle*.

A cycle consists of two steps:

1. Creation of new extensions to the current representations. This may involve creation of new r-elts or transfer of existing r-elts from the STS to the LTS (see section 6.4.1).
2. Sorting and trimming of the extensions. This involves deciding which representations to maintain in the interpretation and which to discard (see section 6.4.2).

6.4.1 Creating New Extensions

If all of the observations in the text have been observed and the incoherence of the best representation is within Tolerance, IDC transfers the contents of the STS to the LTS and halts comprehension, displaying its results. In this case, there is no need to sort and trim representations.

If some statements remain to be observed and the current incoherence of the r-elts in the STS is equal to or below the level of Tolerance, IDC reads in the next observation. This may be merged with existing instances in either memory store, or a new observed instance representing the observation may be generated.

If the current incoherence of the STS is above Tolerance, IDC creates new *extensions* based on its current representations in an effort to lower incoherence. IDC has three methods for creating extensions:

1. *Create new trees*

IDC connects instances into trees, inferring new instances where necessary. As this procedure is the core of IDC, it is described in more detail in the next section.

2. *Transfer trees from the STS to the LTS*

IDC transfers a single tree from the STS to the LTS; simultaneously, any instances which occur only in that tree and not in any other tree are transferred to the LTS. This represents IDC's ability to prioritise the content of the STS when Tolerance is exceeded. Instances which are already part of a tree have been integrated (at least partially), so are less important than instances which are wholly isolated.

3. *Transfer isolated instances from STS to LTS*

If no trees can be transferred from STS, IDC can transfer isolated instances (i.e.

those which occur in no tree) from the STS to the LTS. This is a last resort, as it means that those instances are being ‘put aside’ for integration at a later date. The least incoherent instances are transferred first, as they have the least potential for being connected to the remainder of the text.

IDC uses each method in turn, applying it several times to each of the representations currently constituting the interpretation. Each application of the method to a representation R creates a new extension E ; E contains both a new representation R' derived from R and a value representing the incoherence difference between R' and R ; this is the *incoherence change* (*inc_change*) of the extension. The format of an extension is thus:

$$ext(R', inc_change)$$

If the incoherence change is ≤ 0 , the extension is a viable way for IDC to lower the incoherence of the STS; if the value > 0 , the extension is immediately discarded. When a new tree is created there is no guarantee that incoherence will fall; however, transfers from the STS to the LTS guarantee a drop in incoherence.

Thus, each method either produces a batch of extensions whose (global) incoherence is lower than the representations currently in the interpretation, or no extensions at all (only possible in the case of new tree creation). (Some of the extensions may have a higher incoherence than the original representations, but there is a general trend towards lower incoherence across the interpretation.) If no viable extensions were produced, the next method in the list is tried. This process ensures that transfers are not tried until creation has failed to produce a viable extension; otherwise, information would pass through the STS ‘without touching the sides’.

Once a set of viable extensions has been created, they are sorted with reference to the *Range* parameter. As this has important psychological ramifications, it is described in detail in section 6.4.2.

In the next section, I cover in greater depth the process by which new trees are created.

Creating New Trees

The creation of new trees which connect and integrate nodes is IDC’s principle method for lowering incoherence. All possible extensions based on the creation of trees are tried before r-elts are moved to LTS; this simulates the ‘drive for coherence’ central to comprehension, which encourages integration of inputs over raw storage [Graesser et al., 1994], [van den Broek et al., 1995].

A tree can be constructed either *top-down* or *bottom-up*. In both cases, the process is based on a production system style of programming [Frost, 1986]:

1. Select a ‘trigger instance’ from the STS.
2. Match the trigger instance against an antecedent or consequent node of a schema. (Instances are not literally matched against the data structures representing IDC’s schemas, but are matched against the indexes produced by the *IDC_setup* module.)
3. Retrieve the matched schema.
4. Construct a tree using the schema, with the trigger instance as its first element. The trigger instance may be the tree’s parent or one of its children, depending on the direction in which the tree is being constructed:
 - If the trigger instance matched the antecedent of a schema, the tree is constructed top-down (by forward-chaining).
 - If the trigger instance matched a consequent of a schema, the tree is constructed bottom-up (by backward-chaining).

The rest of the tree is constructed by matching nodes in the schema with instances in the STS; then by matching with instances in the LTS; then by creating new instances to fill any ‘blanks’ in the tree. When an instance is matched with a node in the schema, the constraints on that node are checked.

Note that all of the nodes specified in the schema are added to the representation. Thus, the schema’s contents are being asserted as a single ‘cognitive unit’ (see section 4.2.1).

5. Update the incoherence of instances included in the tree, and/or determine the incoherence of newly-introduced instances. These figures are used to determine the incoherence change generated by construction of the tree.

A detailed example of tree creation is given in appendix A.

When a new tree is added to a representation, IDC checks for r-elts (trees and instances) which have become *unnecessary* as a consequence of the new tree. This procedure is described in more detail in the next section.

Detecting Unnecessary R-elts

Although IDC is capable of maintaining multiple representations (see section 6.3.3), it is also capable of correcting existing representations. This approach differs slightly from the traditional one in AI, where representations are not corrected; instead, they are deleted when their quality falls too far behind the best representation's [Goldman, 1990], [Hobbs et al., 1993], [Ng, 1992].

The model of representation correction in IDC has some similarities with Krems and Johnson's model of anomalous data integration in abduction [Krems and Johnson, 1995]. In their model, inconsistent explanations are deliberately rejected and an attempt is made to re-explain the data that they explained. After re-explanation, the best explanation is retained, whether this is the original explanation or the re-explanation (in this case, the best explanation is the one which 'explains the most data with the fewest number of explanatory components' (ibid.), c.f. simplicity). IDC is slightly different, in that new inferences are generated from a representation to produce new trees which may explain or elaborate existing instances. There is no initial rejection of existing trees. Then the new trees are compared with existing trees (in both memory stores) to determine whether any of them have become (potentially) unnecessary. Whether potentially unnecessary trees are removed depends on whether this causes a reduction in incoherence.

IDC marks existing trees as potentially unnecessary in three situations:

1. *Redundancy by subsumption*

Briefly, an existing tree E may be made redundant by subsumption with respect to a new tree N if the observations explained by E are also explained by N . In more detail, subsumption occurs if the following conditions hold:

- The parent of E is not a child of N . (The reason for this condition is to allow whole trees to be subsumed under a new tree.)
- The observed children of E are a subset or equal to the observed children of N .
- E has higher incoherence than N .

2. *Redundancy by replication*

An existing tree E may be made redundant by replication with respect to a new tree N if both E and N have the same parent but different children. Replication occurs if the following conditions hold:

- The parent of E unifies with the parent of N .¹
- E has higher incoherence than N .

3. *Spuriousness*

Spurious trees tend to result from removal of a tree which has been produced by elaboration from another tree. An existing tree E may be spurious if the following conditions hold:

- E has an inferred parent and no observed children (i.e. E is *groundless*).
- The parent of E does not occur as a child in another tree and no child of E occurs as a parent of another tree (i.e. E is *isolated*).

After removal of a tree, the instances in both memory stores are updated. Their parent and child flags are set, according to whether the instance occurs as a parent or child of any tree which remains in memory. This may result in some inferred instances being both parent- and child-less (i.e. its parent and child flags set to ‘n’). IDC removes any such instances from memory, which again contributes to incoherence change.

The removal of trees in IDC is psychologically unrealistic because IDC has full access to all of the contents of memory. For example, if a tree formed early in comprehension becomes redundant towards the end of comprehension, that tree may be removed with relatively little effort. What is required is some limit on the distance over which necessity checking operates. However, I have been unable to adequately decide this limit: for the purpose of preventing endless useless inferences, the system must be allowed to retrieve and remove trees which are superseded by better ones, and the process must be allowed to occur over arbitrary distances. In versions of the system which do not incorporate tree removal, the system is quickly overcome by multiple, often contradictory explanations of the same events. As there is no mechanism in IDC for consistency checking beyond incoherence change (e.g. no checking for contradictory types assigned to the same entity), tree removal provides the only mechanism for enforcing consistency in representations.

Transferring R-elts to the LTS

As noted in section 6.4, IDC attempts to transfer r-elts from the STS to the LTS if no viable extension(s) can be created by inference. Trees or isolated instances (i.e. instances

¹Strictly speaking, if a copy of E unifies with a copy of N . Copies are used to prevent erroneous changes to the content of instances which are only being tested for redundancy.

which do not participate in any tree) may be transferred.

IDC attempts to transfer trees first, along with their dependent instances. Dependent instances are defined as follows:

An instance is dependent on a tree T in the STS if it occurs only in T and not in any other tree in the STS.

If there are no trees in the STS, instances are transferred. Instances with low incoherence are transferred first.

The transfer process itself is basic, simply involving removing r-elts from the STS of a representation and appending them to the r-elts of the LTS of the same representation. There is no possibility for errors in coding, forgetting, better encoding for more active components [van den Broek et al., 1999], etc..

6.4.2 Updating the Interpretation

IDC's interpretation consists of one or more representations (see section 6.3). After each cycle when comprehension processes are applied (see section 6.4), one or more extensions are returned. IDC now has to decide which of those extensions to maintain as part of its interpretation and which to discard.

To do this, IDC uses a type of *beam search* algorithm to 'sort' and 'trim' the list of representations. The width of the beam employed is defined by IDC's *Range* parameter: the greater the Range, the more representations will (generally) be maintained. If Range is set to -1, only one representation is maintained at the end of each cycle.

The sort and trim algorithm is as follows:

1. Sort extensions in ascending order of incoherence: the extension with the lowest incoherence is placed at the front of the list.

For example, if the extensions list were:

$$[ext(repr(STS1, LTS1, 10.2), -1), ext(repr(STS2, LTS2, 11), -1.5), ext(repr(STS3, LTS3, 10), -1.2)]$$

The sorted extensions list would be:

$$[ext(repr(STS3, LTS3, 10), -1.2), ext(repr(STS1, LTS1, 10.2), -1), ext(repr(STS2, LTS2, 11), -1.5)]$$

Note that incoherence change is not used here, but is maintained for displaying on screen.

2. Select the first extension as the ‘best’ extension B .
3. Set the *total_width* of the interpretation to 0.
4. Set the list of representations to be carried over to the next cycle, C , to $[B]$.
5. For each remaining extension E in the list:
 - (a) Subtract $inc(B)$ from $inc(E)$. The resulting figure is the *divergence* of E .
 - (b) Add the *divergence* of E to *total_width*.
 - (c) If $total_width \leq Range$, add E to C . Otherwise, discard E and return C .

For the example above, the divergence of the second extension in the list = $10.2 - 10 = 0.2$ and the divergence of the third extension is = $11 - 10 = 1$. If $Range$ were set to 0.2, only extensions one and two would be maintained; if it were set to 1.2 or higher, all three extensions would be maintained.

If the divergence of a representation is 0 (with respect to the best representation), it is maintained (even if $Range$ is set to 0).

6.4.3 Psychological Correlates of Range

Range acts in tandem with Tolerance to manage inference generation. The role of Tolerance is to determine when an inference should be attempted; it specifies the amount of information (‘potential structure’) IDC judges as being required before useful inferences are likely to be produced. Range examines the results of inference generation and decides how many alternatives to maintain. However, is there any evidence that human comprehenders are able to perform the same feats?

Most previous work on multiple representations has focused on the sentence level. For example, Just and Carpenter examined whether multiple representations of syntactically-ambiguous sentences are maintained in short working memory span and high span subjects [Just and Carpenter, 1992]. They found that high span subjects had slower reading times on ambiguous sentences than on unambiguous sentences, while low span subjects had equivalent reading times for both. Their hypothesis is that high span subjects

have sufficient capacity to maintain the alternative representations; however, while doing this, they have to manage more information and so are unable to read as quickly as in unambiguous contexts. By contrast, low span subjects maintain a single representation in both the ambiguous and unambiguous contexts; they only have sufficient capacity to maintain one representation. However, Just and Carpenter did not examine alternative representations at the schema level: for example, if presented with an eventuality which could potentially be explained by two higher-level plans, do comprehenders maintain one, both or neither of those plans?

There is little research on maintenance of multiple representations at the schema level. As I showed in section 3.2.5, symbolic-connectionist implementations broach this subject, as the simultaneous activation of nodes representing competing explanations can be viewed as multiple representations. However, the problem is again the lack of executive function in these models: it is difficult to take a single model and change its behaviour without altering the inner workings of its relaxation algorithm (but see section 5.3.6 for Zwaan's possible solution).

Range is intended as a representation of some aspect of the central executive responsible for monitoring the progress of the interpretation as a whole. This differs from Skepticism, which monitors the progress of individual representations and defines the comprehender's attitude towards the utility of inferences. Instead, Range is akin to Skepticism at the level of the interpretation: it represents the comprehender's willingness to explore alternative representations.

Range can be used to test hypotheses about how/if multiple representations are maintained by human comprehenders without having to change the details of the knowledge base or the quality metric. In IDC, maintenance of a single representation is efficient for texts which are not misleading, or even predictable: because comprehenders' inferences tend to be confirmed by future statements in the text, there is no need for retraction of incorrect inferences. However, where texts are misleading or ambiguous, maintenance of multiple representations may be more efficient: the comprehender may have a preference for one representation over the others, but retain alternatives in case they are useful later. If the preferred representation fails to account for a statement in the text, the comprehender can switch it with one of the alternatives which *is* able to account for it. However, as multiple representations may be maintained throughout comprehension, the storage capacity required is higher than in the single representation condition.

One area where differences between maintenance of single and multiple representations

may manifest is during comprehension of role-shift texts [Sanford and Garrod, 1981]. I have carried out some experiments with IDC on such texts, and present some predictions about human behaviour on the basis of these experiments in the next chapter.

6.5 Chapter Summary

This chapter gives a high level account of the processes used by IDC. The emphasis of the system is the continual assessment and management of extensions to the interpretation. The key inferential control processes can be divided into three groups:

1. *Controls on which inferences are possible*

These limits are largely due to technical details of the implementation:

- (a) No inferences beyond the available schemas can be made.
- (b) No inferences can be triggered by a relation node. There are technical reasons for this: it prevents two events being accidentally assigned the same token, e.g. it prevents two distinct events from both having the token *e1*.
- (c) No inferences can be made which violate variable-binding restrictions. While this is largely for technical reasons, there may also be a grain of psychological truth in this: for example, this prevents inference of a *stabbing* event from a single observation of someone holding a *knife*.

2. *Controls on which inferences are useful*

These constitute important theoretical limits, relying as they do on examining Tolerance and the incoherence change created by inferences:

- (a) Tolerance determines *when* to make inferences, and when enough have been made.
- (b) Skepticism determines *which* inferences are most useful by modulating the incoherence change created by an inference.

3. *Controls on which inferences are maintained*

Range governs which representations are considered ‘good enough’ to be retained for further investigation.

Of these three forms of control, the first type is the most psychologically suspect. The other two types may approximate some mechanisms of human comprehension: Skepticism (see section 5.3.6), Tolerance (see section 6.3.1), and Range (see section 6.4.3) have been discussed in this capacity.

In the next chapter, I explore how these processes operate in various contexts.

Chapter 7

Examples of IDC's Behaviour

I have used IDC to model several types of text comprehension:

1. *Plan Recognition*

As the program was designed primarily as an abductive system, plan recognition is naturally a suitable task for it. This strand of my work follows in the tradition of [Ng and Mooney, 1990], [Ng, 1992], [Hobbs et al., 1993], and [Charniak and Goldman, 1993].

2. *Inference Protocols*

Much of the development of the implementation took place within the context of a simulation of verbal protocols described in [Trabasso and Magliano, 1996]. As a result, the metrics were designed to cope with this and similar texts, producing an inference trace similar to that of human comprehenders.

3. *Role-Shift Texts*

The initial prototype of IDC only had facilities for maintenance of a single representation. However, members of my thesis group suggested that multiple representations may be necessary in cases where texts are ambiguous; I incorporated this suggestion into the final version of the implementation. One useful aspect of this is that hypotheses about the maintenance of multiple representations can be generated on the basis of IDC's behaviour. The effect of multiple representations may be most pronounced and accessible during comprehension of particularly awkward, deliberately-misleading texts, known as *role-shift* texts in the literature [Sanford and Garrod, 1981]. Here, the comprehender may be led to make one assumption about the roles of the characters involved in a text, only to find that

these assumptions must be revised and replaced later in the text. IDC's behaviour demonstrates two potential mechanisms for dealing with such corrections: assertion of new trees and deletion of existing trees; or simultaneous maintenance of alternative representations until one becomes sufficiently more determined than the others, at which point the others can be discarded.

In the rest of this chapter, I describe each of these three areas of application, giving examples of IDC's comprehension of different kinds of text.

7.1 Plan Recognition

IDC can be equipped with schemas in the style of traditional plan recognition systems; many of the examples earlier in this thesis were based on such schemas. The schemas I have used in IDC for plan recognition are based on those of ACCEL [Ng, 1992]. I have used ACCEL as a basis, partly because the work is closest to my own intentions and partly because the code is freely available for scrutiny.¹ Ng's thesis also contains a definitive set of plan recognition texts, some of which I have used for testing IDC.

The main difference between ACCEL's rules and IDC's schemas is that ACCEL uses Horn clauses (rules with a single consequent node), while IDC uses schemas with a single antecedent node. As a result, IDC's schemas are an amalgamation and merging of several ACCEL rules. For example, consider the ACCEL rules:

```
(← (inst ?g going) (inst ?s shopping) (go-step ?s ?g))
(← (goer ?g ?p) (inst ?s shopping) (go-step ?s ?g) (shopper ?s ?p))
(← (dest-go ?g ?str) (inst ?s shopping) (go-step ?s ?g)
    (store ?s ?str))
(← (inst ?sp shopping-place) (inst ?s shopping) (store ?s ?sp))
```

These rules have the form:

$$(\leftarrow \textit{consequent} \textit{antecedent}_1, \dots, \textit{antecedent}_n)$$

The rules may be manually converted into the partial schemas:

¹from <ftp://ftp.cs.utexas.edu/pub/mooney/accel>.

```

event(S, shopping, [shopper:P, store:S, ...]) / [] --->
  [
    event(G, going, [agt:P, loc_to:S, ...]) / [],
    habit(shopping_place, [exp:S]) / [],
    ...
  ].

habit(store, [exp:S1]) / [S1 ==> S2] --->
  [
    habit(shopping_place, [exp:S2]) / []
  ].

```

'...' stands for other events and roles which could be added to a schema, as each schema integrates information from several ACCEL rules into a single unit.

Note that a constraint has been added to the second schema: $[S1 ==> S2]$. This ensures that a vague inference is not made from any kind of *shopping_place* to the more specific *store*. This technique is used throughout IDC's *isa* hierarchy to limit inferences from less-specific to more specific nodes (remember that $--->$ represents a generalisation relationship, with the antecedent node being a more specific description of the consequent nodes). Such inferences are possible if instances of both nodes in the schema already exist in the representation; however, if there is only an instance of the more general node (the consequent), inference to the specific node is prevented.

(Also note that I have not retained the *go-step* of Ng's rule. This is because 'step' eventualities are not introduced unless they are required to discriminate two different means of instantiating a step within a plan. For example, if there were several different ways for *go-step* to be instantiated (e.g. going by bus, by taxi, on foot), the *go-step* node would spawn a schema for each alternative. However, until the schemas which define these alternatives are added to the knowledge base, it is assumed that there is no need to discriminate types of 'going'.)

Using the conversion scheme described above, it is possible to derive a set of IDC schemas capable of deriving results similar to those of ACCEL. A listing of the schemas is given in section B.1.1 (page 222).

The texts analysed by these schemas are translations of those analysed by ACCEL, such as:

Bill got a gun. He went to the supermarket.

The translation of this text into a format readable by IDC is:

[event(e1, get, [agt:bill, pat:w, _]), habit(gun, [exp:w]), event(e2, go, [agt:bill, loc_to:sm]), habit(supermarket, [exp:sm])]

Note that no temporal information is used in plan descriptions or in texts. This is because ACCEL does not use temporal relations, and because of unresolved issues concerning relations in IDC (see section 8.2.1).

Evaluation of ACCEL compared the program's output with the author's intuitions about the best interpretation for a text. In the case of IDC, I was more interested in showing how interpretations could vary according to different parameter settings and the time-course of comprehension; this means that the correctness of interpretations is a less important criterion. However, at 'average' parameter settings, IDC's outputs at the end of comprehension are close to those of ACCEL.

An example of IDC's performance on plan recognition texts is given in the following section.

7.1.1 An Illustrative Example

Consider the following example, a 'classic' in the plan recognition literature:

Bob went to the liquor-store. He pointed a gun at the owner.

The IDC version of this text is:

[event(e1, go, [agt:bob, loc_to:ls]), habit(liquor_store, [exp:ls]), event(e2, point, [agt:bob, pat:w, obj:o]), habit(gun, [exp:w]), habit(owns, [agt:o, pat:ls])]

What kinds of behaviour are possible, given this text? According to both Goldman and Ng [Goldman, 1990], [Ng, 1992], after reading the first sentence, the obvious explanation is that Bob is going shopping for liquor. Then, after reading the second sentence, the explanation shifts, so that Bob is robbing the liquor store. This is the order in which these explanations are formed in both of their systems.

However, as I've tried to make clear throughout this thesis, it is not necessarily the case that an explanation will be formed on the basis of 'Bob went to the liquor-store' alone. Comprehenders' construction of this explanation relies on several factors, including:

1. The amount of attention they are focusing on the text.
2. The amount of control they have over their inference processes.
3. The capacity of their working memory.

These are the factors which are modelled in IDC, and which mean that the above explanations are not necessarily inferred after comprehending the first sentence of the text. On the next page, the results for various runs of IDC with different parameter settings are shown (table 7.1). The results are shown in English (rather than Prolog) to make them more concise. For the sake of completeness, examples of the actual trees constructed by IDC are included in section B.1.2 (page 229).

Table 7.1 shows the program's output for three settings of Tolerance, but not for alternative settings of Skepticism and Range. This is because there are a limited number of potential representations for this text, so the results are the same for any setting of Range. Similarly, Skepticism makes no difference to the rate at which inferences are made, as only one representation is possible; as a result, the above results are true for cases where $0 < \text{Skepticism} \leq 0.5$. However, when Skepticism is over 0.5 no inferences are produced *at all*. This is because the text supplies few cues (with respect to the schema lattice) which uniquely determine an appropriate schema (with respect to $\text{Skepticism} > 0.5$).

The entries in the table show the particular operation for each comprehension cycle (see section 6.4). 'Observed' means that the next statement in the text was read into the STS (possibly merged with existing parts of representations); 'Inferred' means that a new tree was inferred (see section B.1.2 for details of the exact content of trees); 'Transferred' means that a tree was moved from the STS to the LTS.

There are several interesting points to note:

- The initial inference of *robbing* in the first two cases (Tolerance = 1, Tolerance = 2) refers only to some location, not to the liquor-store. This is because the *robbing* schema contains a `==>` constraint which prevents *ls* being bound as the place robbed when the schema is initially applied (shown in bold below):

```
event(_, robbing, [robber:A, weapon_used:W, place_robbed:P2,
thing_robbed:V, victim:M]) / [V \== W] --->
[
  event(_, get, [agt:A, pat:W, _]) / [],
```

Tolerance	Cycle(s)	Operation
1	1	Observed: Bob going to <i>ls</i> .
	2	Inferred: Bob's going isa go-step.
	3	Transfer: Bob's going isa go-step.
	4	Observed: <i>ls</i> is of type 'liquor-store'.
	5	Observed: Bob pointed <i>w</i> at <i>o</i> .
	6	Inferred: 'robbing' event - Bob robbed some place using the weapon <i>w</i> , with <i>o</i> the victim.
	7	Observed: <i>w</i> is of type 'gun'.
	8	Inferred: <i>w</i> is of type 'weapon' (integrates <i>w</i> with 'robbing' event).
	9	Transfer: 'robbing' event.
	10	Observed: <i>o</i> owns <i>ls</i> .
	11	Inferred: <i>o</i> is in charge of <i>ls</i> (integrates <i>o</i> 's ownership of the store with the robbing event, and binds <i>ls</i> as its location).
	12-14	Transfer: trees in the STS transferred to the LTS.
2	1	Observed: Bob going to <i>ls</i> .
	2	Observed: <i>ls</i> is of type 'liquor-store'.
	3	Inferred: Bob's going isa go-step.
	4	Observed: Bob pointed <i>w</i> at <i>o</i> .
	5	Inferred: 'robbing' event - Bob robbed some place using the weapon <i>w</i> , with <i>o</i> the victim.
	6	Transfer: 'robbing' event.
	7	Observed: <i>w</i> is of type 'gun'.
	8	Inferred: <i>w</i> is of type 'weapon' (integrates <i>w</i> with 'robbing' event).
	9	Observed: <i>o</i> owns <i>ls</i> .
	10	Inferred: <i>o</i> is in charge of <i>ls</i> (integrates <i>o</i> 's ownership of the store with the robbing event, and binds <i>ls</i> as its location).
	11-14	Transfer: remaining trees transferred to the LTS.
3	1	Observed: Bob going to <i>ls</i> .
	2	Observed: <i>ls</i> is of type 'liquor-store'.
	3	Observed: Bob pointed <i>w</i> at <i>o</i> .
	4	Observed: <i>w</i> is of type 'gun'.
	5	Inferred: <i>w</i> is of type 'weapon'.
	6	Observed: <i>o</i> owns <i>ls</i> .
	7	Inferred: <i>o</i> is in charge of <i>ls</i> .
	8	Inferred: Bob's going isa go-step.
	9	Inferred: 'robbing' event - Bob robbed <i>ls</i> using the weapon <i>w</i> , with <i>o</i> the victim.
	10-14	Transfer: remaining trees transferred to the LTS.

Table 7.1: Plan recognition inferences: operations during comprehension of the liquor-store text.

```

event(_, go_step, [agt:A, loc_to:P2]) / [],
event(_, point, [agt:A, pat:W, obj:M]) / [],
event(_, get, [agt:A, pat:V, from:M]) / [],
habit(valuable, [exp:V]) / [],
habit(business, [exp:P1]) / [P1 ==> P2],
habit(weapon, [exp:W]) / [],
habit(in_charge, [agt:M, pat:P2]) / []
].

```

However, the constraint does ensure that if any of the elements later bind the location to a constant, the changes are propagated throughout the tree. Thus, when the inference is made that the owner of *ls* is in charge of *ls* (cycle 10 when Tolerance = 1, and cycle 11 when Tolerance = 2), the binding is propagated through all of the trees, allowing the liquor-store *ls* to become the location where the robbing takes place.

- When Tolerance < 3, the *go_step* inferred from the *going* event is not recognised as being the same *go_step* as that of the *robbing* event. This is because of the constraint mentioned in the previous paragraph, which prevents the *loc_to* of the *go_step* of *robbing* being bound before the *business* child's *exp* role.

However, when Tolerance = 3, the *go_step* is inferred before the *robbing* event. This means that the *going* event with Bob as its agent is recognised as constituting the *go_step* of the robbery.

This sequence of inferences is slightly arbitrary: the lack of integration between the inferred *go_step* and the *robbing* event is partly due to the way IDC compares and matches events when building trees. However, there is the interesting possibility that the failure to connect the events is due to the inaccessibility of the previous inference. It may be that human comprehenders fail to revise inaccessible parts of their representations due to a similar problem: once elements of the representation are lost from the STS, they are less available for revision and possible connections to them may be missed [Johnson and Seifert, 1999].

- A general point is that as Tolerance increases, the efficiency of comprehension also increases. At high Tolerance, there is less time wasted on moving r-elts between the

two stores, as more can be maintained simultaneously in the STS. This means that inferences are made in a continuous stream (cycles 7-9), efficiently tying together the r-elts in the STS without the need for retrieval and transfer.

One criticism might be the tendency for a large number of r-elts to be maintained in the STS when Tolerance is greater than 1. This number far exceeds the limits suggested by various researchers, e.g. [Fletcher et al., 1990], [Trabasso and Magliano, 1996]. However, Ericsson and Kintsch's Long-Term Working Memory (LT-WM) theory [Ericsson and Kintsch, 1995] suggests that the number of elements readily available during comprehension is greater than the relatively small 5-9 adhered to by many researchers (as a result of Miller's work [Miller, 1956]):

...links between propositions currently in the focus of attention and propositions in the long-term episodic text memory, which are established incidentally by the very nature of the comprehension process, make available to the reader a large subset of the text memory in LTM, thus generating what we call LT-WM. [Ericsson and Kintsch, 1995]

In addition, it is difficult to produce computational models with a very limited capacity [Anderson, 1983]. The inferences suggested by psychologists are at a high level of abstraction, and require various subsidiary inferences to make sense computationally (see section 7.2). With a very small STS capacity, it is hard to manipulate sufficient data for such detailed inferences.

Table 7.1 demonstrates how increasing Tolerance delays the inference process until more information is in the STS. However, the end result is almost the same for all three settings of Tolerance, as the inferences which are actually made and accepted depend not on Tolerance, but on Skepticism and Range. In this case, as I explained previously, the lack of alternative explanations means that Skepticism and Range exert no influence over comprehension of the text. In the next two sections, I describe some changes to the knowledge base which *do* influence comprehension.

Adding Other Schemas

The first alteration I made to the knowledge base was the addition of a new schema:

```

event(_, mugging, [mugger:A, weapon_used:W, thing_robbed:V,
victim:M]) / [] --->
    [
    event(_, point, [agt:A, pat:W, obj:M]) / [],
    event(_, get, [agt:A, pat:V, from:M]) / [],
    habit(valuable, [exp:V]) / [],
    habit(weapon, [exp:W]) / []
    ].

```

This changes comprehension because it provides an alternative explanation for Bob pointing a gun at the owner of a liquor store (i.e. Bob is mugging the owner).

By adding a schema for *mugging* events which has a *point* node as one of its consequents, IDC has two alternative representations for the text (assuming the constraint in the *liquor_store_shopping* schema has not been removed):

1. A representation where Bob is mugging the owner of the liquor store.
2. A representation where Bob is robbing the liquor store.

Is inference of a *mugging* event valid in this context? If the *robbing* and *mugging* schemas are compared, it is clear that *robbing* trees have more children than *mugging* ones. Therefore, given that IDC prefers to assume as few new nodes as possible, the *mugging* representation is preferred. More of the *robbing* schema's nodes are instantiated by the text than are the *mugging* schema's nodes; however, there are fewer of the *mugging* schema's nodes left uninstantiated by the text than there are of the *robbing* schema.

The addition of this schema changes the behaviour of IDC, depending on the settings of the Skepticism and Range parameters. I will not go into any great detail, but merely note the following:

- If Skepticism is low (≈ 0.1), the tendency is for two representations to be constructed, one representing the *robbing* explanation and one the *mugging* explanation. Depending on the setting for Range, both may be maintained, or only the least incoherent (the *mugging* explanation).
- If Skepticism is higher (≈ 0.5), IDC refuses to produce any inferences. This is reasonable behaviour, given that the support for either representation with respect to the knowledge base is fairly slim. However, there is a problem here, as *pointing*

events in combination with the presence of objects of type *gun* should provide a good cue about which schemas to apply. The fact that no inferences are made seems to contradict our intuitions about the correct representation, that Bob is robbing the liquor store.

A remaining question is whether the *mugging* representation is valid at all in an indoor context. Do muggings occur indoors? Or, by definition, do muggings occur outside buildings? Is there perhaps a way to block the *mugging* representation on this basis? Perhaps if the conditions under which muggings take place were specified more clearly in the schema, the difference in incoherence between the representations derived from the two schemas may be lessened; it may even be the case that the *robbing* representation would have lower incoherence. However, without a subjective judgement of which schema should 'win' the competition, for every possible combination of conditions, it is impossible to decide on these conditions.

Removing Constraints

The second change I made to the knowledge base was to remove the constraint which prevents *liquor_store_shopping* being inferred. The unaltered schema is:

```
event(_, liquor_store_shopping, [shopper:S, store:T2,
thing_bought:B2]) / [] --->
  [
    event(_, shopping, [shopper:S, store:T2,
    thing_bought:B2]) / [],
    habit(liquor_store, [exp:T1]) / [T1 ==> T2],
    habit(liquor, [exp:B1]) / [nonvar(B1), B1 ==> B2]
  ].
```

I removed the constraint shown in bold. Normally, this prevents inference of a *liquor_store_shopping* event unless there is an explicit mention of a *liquor* instance in the representation. In other words, the instance instantiating the *habit(liquor, [exp:B1])* node must bind the variable *B1* to some constant, e.g. *x*; this binding is then passed to the *shopping* event, and then up to the *liquor_store_shopping* event.

By removing the constraint, a *liquor_store_shopping* event may be inferred without explicit mention of a *liquor* entity; the actual effect again depends on Skepticism.

With low Skepticism, both the *liquor_store_shopping* and the *robbing* trees are added to the representation. This is because the *robbing* tree makes no explicit reference to a *liquor_store*: any *business* can be instantiate the *place_robbed* role value. By contrast, the *liquor_store_shopping* tree incorporates the *liquor_store* of the text directly, with no intermediate inferences; it is thus a much 'purer' method of integrating *liquor_store* into the representation than the *robbing* tree, which requires an inference from *liquor_store* to *business*.

If Skepticism is around 0.5, the representations will not change: the derived representations are as those of table 7.1. The reason for this is that the constrained *liquor_store_shopping* schema still exerts an influence over whether a *robbing* event is inferred, even though it will not be inferred itself. This is because the *liquor_store_shopping* schema alters the informativity and ubiquity of the nodes in the *robbing* event, even though there is no direct overlap. Instead, the effect occurs because both the *shopping* schema and the *robbing* schema contain a *go_step* node; and the *liquor_store_shopping* schema contains a *shopping* node; thus, the ubiquity of the *shopping* node is increased; and this has the knock-on effect of increasing the ubiquity of the *go_step* node.

Consequently, if the constraint is removed, it makes no difference to whether IDC with Skepticism = 0.5 infers a *liquor_store_shopping* event: the incoherence of *any* inference is too high. This shows clearly how constraints exert control over inference generation (see section 6.5).

7.1.2 General Discussion

IDC can simulate the plan recognition behaviour typical of abductive systems. However, it is difficult to evaluate IDC using the evaluation criteria of these systems. IDC is inherently unstable, in that changes to the knowledge base can irrevocably change the system's behaviour. This makes measures such as precision and recall unreliable as indicators of the system's capabilities [Ng, 1992].

This is not necessarily a damaging criticism. The strength in IDC is its demonstration of how changes to the knowledge base have a profound effect on representation quality measurement. This point is not often made in connection with quality metrics based on structural criteria, such as simplicity (see section 5.1.2) and ACCEL's coherence metric (see section 5.2.4). However, in such metrics, alterations to the knowledge base may shift a representation's quality from 'good' to 'bad' (or vice versa). These metrics are successful by virtue of the way the rules are structured; changes to these structures change the

representations which are possible, and thus the measurement of quality.

In IDC, the instability of comprehension (with respect to the knowledge base) is precisely what I intended to model when designing the metric. However, getting the balance right between nodes which allow discrimination between schemas (i.e. nodes which occur infrequently) and nodes which actually trigger tree construction (i.e. nodes which occur frequently) is very complicated. Too often, a small change to the knowledge base can have drastic (and/or disastrous) effects. In this respect, the metric is more sensitive to slight changes (e.g. addition of a constraint) than I would have liked.

In other respects, if the knowledge base remains constant and is reasonably well-designed (e.g. through extensive trial and error), it is possible to produce inference protocols which resemble those of human comprehenders. In the next section, I describe the inference protocol which IDC produces from longer texts which are not wholly plan-based.

7.2 Inference Protocols

Verbal protocol methods have for several years been used to uncover the inference processes of human comprehenders [Trabasso and Magliano, 1996]. Such protocols often involve question answering and/or 'think aloud' protocols. In the latter category is the work described in [Trabasso and Magliano, 1996]. They analysed some think aloud protocols gathered by Suh, where each protocol was produced by a comprehender who was told to communicate their understanding of the text to the researcher (Suh) [Suh and Trabasso, 1993]. Trabasso and Magliano's aim was to specify the memory operations and inference processes of comprehension. To do this, they parsed the protocols into clauses, and annotated each clause with the inference operation which produced it and the memory operation it involved. As the clauses were produced *during* reading, they could be aligned with the sentences which engendered them.

The resulting parsed protocols are some of the most detailed descriptions of on-line inference behaviour I have discovered in the cognitive psychology literature. For this reason, they proved useful when designing IDC: I took a protocol (from [Trabasso and Magliano, 1996]) and devised a set of schemas which could conceivably produce it. The text on which the protocol is based is the 'Ivan Story':

- S1. Ivan was a great warrior.
- S2. Ivan was the best archer in his village.

S3. Ivan heard that a giant was terrifying the people in his village.

S4. The giant came to the village at night and hurt people.

S5. Ivan was determined to kill the giant.

S6. Ivan waited until dark.

S7. The giant came and Ivan shot an arrow at him.

S8. Ivan hit the giant and the giant fell down.

S9. The people were overjoyed.

(adapted from Trabasso and Magliano [Trabasso and Magliano, 1996])

The protocol associated with this text is presented by Trabasso and Magliano sentence by sentence, as follows:

Sentence	Clauses	Inference Operation	Memory Operation
S5. Ivan was determined to kill the giant.	c1. As expected c2. Ivan being a warrior c3. and caring about people c4. will want to kill the giant.	explanation explanation	retrieval activation paraphrase

Table 7.2: A fragment of a text comprehension protocol (from [Trabasso and Magliano, 1996]).

They also represent the protocol using a causal network notation; my own adaptation of this diagram, incorporating the actual content of each clause, is shown in figure 7.2 (page 170).

To construct the schemas, I used the protocols in tandem with my translation of the text into IDC-readable form. For each statement in the IDC-readable text, I looked at the corresponding sentence in the protocol and the clauses it was aligned with; I then attempted to translate the corresponding causes into IDC format; then I designed a schema which would allow the clauses to be derived from the IDC statement. In addition, I used Trabasso and Magliano's network representation of the protocol as a basis for determining causal relations (see the next section). For example, the crux of the clauses in the above protocol fragment is as follows:

- Ivan cares about people; specifically, he cares about the people being hurt by the giant.

- Ivan cares about the people being hurt by the giant because they are villagers from his village.
- In itself, living in the same village as someone is not sufficient motivation for caring about them; presumably, the comprehender makes an inference about them being friends or acquaintances as a result of living in the same village.
- Because Ivan cares about the people in his village, he wants to stop them being hurt; one way of doing this is to disable the enemy which is hurting them.
- Because Ivan is a warrior, one way he has of disabling enemies is to kill the source of the threat; in this case, the giant.

It is clear that the protocols do not contain the various micro-level inferences which are necessary for even the most superficial analysis. By this, I mean that the pair of clauses 'Ivan being a warrior' and 'Ivan caring about people' do not explain 'Ivan wants to kill the giant'. If one were to construct a schema representing this relationship, it might look something like:

$$great_warrior(X), cares_about(X, Y) \longrightarrow goal(X, kill(Z)).$$

This is clearly inadequate as a computational description of the relationships between X caring about some entity Y and wanting to kill some other entity Z as a result (reading \longrightarrow as a causal relationship).

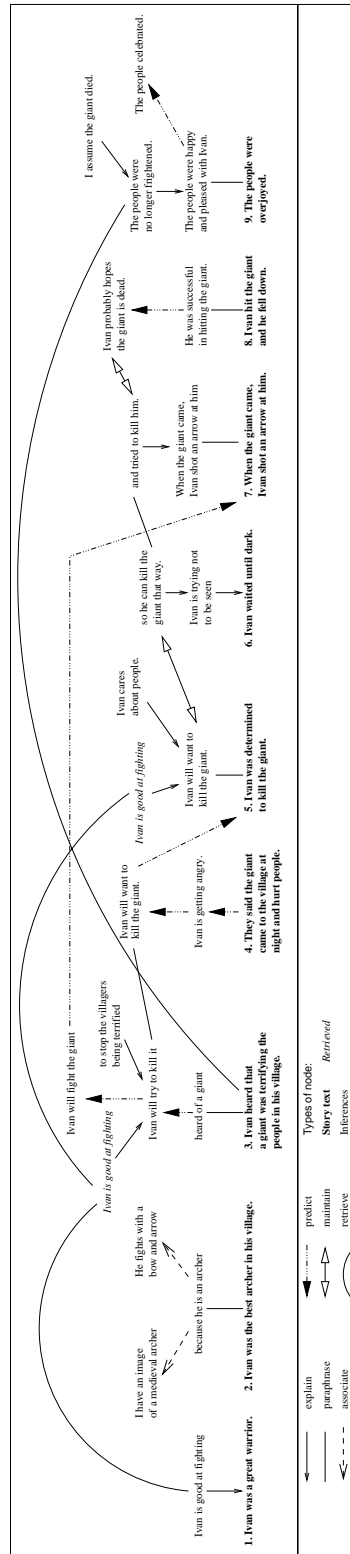


Figure 7.1: Adapted Ivan Story protocol diagram (after [Trabasso and Magliano, 1996])

The human protocol is spartan because it was delivered to another human being; therefore, the comprehender was engaging in a dialogue with the researcher and followed Grice's conversational maxim of Quantity: they only communicated the minimum information necessary about their understanding, expecting the researcher to make inferences from this information [Grice, 1975]. In other words, the comprehender's understanding is a narrative which they expect the researcher to comprehend, making inferences where required. In this light, a reported inference such as '*Ivan is going to try not to be seen to try to kill the giant that way*' should not be taken at face value. It is obvious that the comprehender is trying to communicate the idea of surprise attack or ambush: there is no direct causal connection between being hidden and being better able to attack an enemy.

What is required, then, is a chain of inference resembling the one given in the bullet points above. This involves far more nodes than are suggested by Trabasso and Magliano: for example, after adding the auxiliary nodes, the final representation produced by IDC contains more instances than the representation given by Trabasso and Magliano (52 distinct instances (for Skept = 0.1, Range = -1) vs. 18 respectively).

In the next section, I describe the representational scheme used to design the schemas to produce inferences at the required level of detail.

7.2.1 Making Sense of the Protocol

The representational scheme I required had to allow generation of the same *types* of inference as the inference protocol, but at a level of detail that was sensible in computational terms (see previous section). As the inference protocol contained many causal descriptions, I chose as a starting point the causal network theories of van den Broek and his co-workers [Trabasso and van den Broek, 1985], [Trabasso et al., 1989], [Trabasso et al., 1995], [Trabasso and Magliano, 1996], [van den Broek, 1990b], [van den Broek, 1990a], [van den Broek, 1994]. According to their scheme, a comprehender's representation is dominated by causal relations between instances. Four types of causal relation are specified:

1. *motivates*

A motivates B if:

- *A* is temporally prior to *B*
- *A* is operative when *B* occurs (i.e. *A* exerts some influence on events in the text)

- A is necessary for B (i.e. B would not have occurred in the circumstances of the text if A had not occurred)
- A contains goal information

2. *psychologically causes*

A *psychologically causes* B if:

- A is temporally prior to B
- A is operative when B occurs
- A is necessary for B
- A implies emotion or a cognitive state

3. *physically causes*

A *physically causes* B if:

- A is temporally prior to B
- A is operative when B occurs
- A is necessary for B
- A is sufficient for B (i.e. B is likely to occur in the circumstances of the story if A occurs)

4. *enables*

A *enables* B if:

- A is temporally prior to B
- A is operative when B occurs
- A is necessary for B

Because of the structure of schemas in IDC and the emphasis on ‘encapsulation’, I decided to treat relations between nodes as nodes themselves. A causal relation then becomes a method for specifying the possible representations space for a pair of events: a representation where they occur as a pair is actualised, rather than one where they occur as separate events (see section 5.3.1). (There are some problems with this approach, which I will cover in section 8.2.1.)

For the purpose of writing schemas, it was necessary to analyse the relationships between elements of the protocol, and those between the protocol and the text. This analysis was carried out by hand as consistently as possible, following the specification of causal relations given above and including the auxiliary inferences which would make sense of the protocol. The resulting schemas include some which could be used to comprehend the fragment in table 7.2:

```
habit(care_about, [exp:A2, exp:B2]) / [A2 \== B2] ---->
[
  habit(live, [exp:A1, loc_in:L]) / [A1 ==> A2],
  habit(live, [exp:B1, loc_in:L]) / [B1 ==> B2]
].

relation(psycs, [E2, G2]) / [] ---->
[
  event(E1, terrorise, [agt:T, pat:P]) / [E1 ==> E2],
  goal(G1, kill, [agt:A, pat:T]) / [G1 ==> G2],
  habit(care_about, [exp:A, exp:P]) / [],
  relation(overlaps, [E2, G2]) / []
].
```

In the second schema, *psycs* indicates a relation of type *psychologically causes*: in other words, if *T* terrorises *P*, *A* cares about *P*, and *A* has the goal of *killing T*, then the *terrorise* event *psychologically causes* the *kill* goal. Note that this schema defines a relationship between four sub-nodes, and can thus be used in several ways: if one of the sub-nodes occurs in the text, IDC can backward chain to infer a causal antecedent or causal consequent (see figure 3.4 on page 40); alternatively, if all four sub-nodes occur in the text, the relation may be inferred as a means of connecting those nodes.

As well as the causal schemas shown above, IDC also has script-level schemas similar to those of section 7.1. These define larger event sequences, such as ambushes and heroes protecting victims. The full set of schemas for the inference protocol is included in appendix B.2.

In the following section, I compare the time course and content of IDC's comprehension of the Ivan Story with the verbal protocol referred to in section 7.2.

7.2.2 Time Course and Content of Comprehension

Table 7.3 (below and on the next two pages) shows the results of a run of IDC on the Ivan Story. Each row represents a single comprehension cycle and shows the action taken by IDC during the cycle. The results shown were produced with the parameters set as follows: Skept = 0.5, Range = -1 (i.e. only one representation is maintained), and Tolerance = 0 (i.e. no information is buffered and every observation prompts an attempt to lower incoherence).

It might seem strange to set Tolerance to 0, as I've previously stated that Tolerance embodies the capacity of the short-term store (see section 6.3.1 on page 141). With Tolerance set to 2 and the other parameters at the same values, the same inferences are produced, but the order in which they are generated is different; they also tend to occur in bunches, rather than in smooth progression as they do in the human protocol.

It may be that on-line inferences do occur in groups, at clause and/or sentence boundaries; however, this is not reflected in the talk-aloud protocols because comprehenders were specifically instructed to demonstrate their understanding. This goal actually alters the comprehension process, meaning that the inferences produced do not necessarily mirror inferences routinely produced on-line [Graesser et al., 1996]. As Tolerance is a drive towards integration of r-elts, I simulated the requirement to show understanding by setting Tolerance so that inferences are made at the end of each observation. In other words, Tolerance is set to 0, which forces inferences to be tried after each observation is added to the representation(s). The result is that any reductions in incoherence are tried immediately, rather than observations being 'stock-piled'.

Cycle	Action
1	observed: habit(great_warrior, [exp:ivan])
2	associative inference: <i>Ivan is a great warrior because he is a good fighter</i>
3	transfer: tree from 2
4	observed: habit(best_archer, [exp:ivan, loc_in:village])
5	associative inference: <i>Ivan is an archer</i>
6	associative inference: <i>Ivan uses armour, a bow and arrows, because he is an archer</i>
7	associative inference: <i>arrows are a kind of weapon</i>
8	transfer: tree from 4
9	transfer: tree from 6
10	transfer: tree from 7
11	observed: habit(live, [exp:ivan, loc_in:village])
12	predictive inference: <i>Ivan cares about the others who live in his village</i>

continued on next page

Cycle	Action
13	transfer: tree from 12
14	observed: event(e1, terrorise, [agt:giant, pat:people])
15	associative inference: <i>the giant's terrorising the people constitutes an attack on the people</i>
16	predictive inference: <i>the giant's terrorising the people psychologically causes Ivan to be angry</i>
17	transfer: tree from 15
18	transfer: tree from 16
19	observed: habit(live, [exp:people, loc_in:village]) (merged with prediction from 12)
20	observed: event(e2, come, [agt:giant, loc_to:village, when:night])
21	predictive inference: <i>the giant's coming physically causes some hurting event, with the villagers as its patient</i>
22	transfer: tree from 21
23	observed: event(e3, hurt, [agt:giant, pat:people, inst:_712]) (merged with prediction from 21)
24	observed: relation(precedes, [e2,e3]) (merged with prediction from 21)
25	observed: goal(g1, kill, [agt:ivan, pat:giant])
26	explanatory inference: <i>the giant's terrorising the people psychologically causes Ivan's goal to kill the giant</i>
27	transfer: tree from 26
28	observed: relation(overlaps, [e3, g1])
29	explanatory inference: <i>the giant's hurting the people psychologically causes Ivan's goal to kill the giant</i>
30	transfer: tree from 29
31	observed: event(e4, wait_for, [agt:ivan, pat:giant, loc_at:village, when:night])
32	explanatory inference: <i>the giant's coming to the village enables Ivan's waiting</i>
33	transfer: tree from 32
34	observed: relation(overlaps, [g1, e4])
35	transfer: instance from 24
36	transfer: instance from 34
37	observed: event(e5, arrive, [agt:giant, loc_at:village, when:night])
38	transfer: instance from 37
39	observed: relation(overlaps, [e4, e5])
40	transfer: instance from 39
41	observed: event(e6, shot, [agt:ivan, pat:giant, inst:a])
42	explanatory inference: <i>Ivan's goal to kill the giant motivates his shooting the giant</i>
43	transfer: tree from 42
44	observed: habit(arrows,[exp:a]) (merged with prediction from cycle 6)
45	observed: relation(precedes, [e5, e6])
46	transfer: instance from 44

continued on next page

Cycle	Action
47	transfer: instance from 45
48	observed: event(e7, hit, [agt:ivan, pat:giant, inst:a])
49	explanatory inference: <i>shooting the giant physically causes hitting the giant</i>
50	transfer: tree from 49
51	observed: relation(precedes, [e6, e7]) (merged with tree from cycle 49)
52	observed: event(e8, fall, [agt:giant, loc_to:ground])
53	explanatory inference: <i>Ivan killed the giant by shooting him and hitting him</i>
54	transfer: tree from 53
55	observed: relation(precedes, [e7, e8]) (merged with tree from cycle 53)
56	observed: event(e9, overjoy, [agt:people])
57	transfer: instance from 55
58	transfer: instance from 56
59	observed: relation(precedes, [e8, e9])
60	transfer: instance from 59

Table 7.3: Protocol for IDC's comprehension of the Ivan Story

The **Action** column shows three types of action taken by IDC, corresponding to the processes described in section 6.4 (page 145): *observe* actions take a text statement and add it to the current representation(s); *transfer* actions move either (a) a tree and its dependent instances; or (b) instances alone, from the STS to the LTS; *inference* actions infer a new tree and instances on the basis of an instance in the STS. In the table, I have marked each inference action according to whether it is *explanatory*, *predictive*, or *associative*. IDC does not produce these categories, as it treats all inferences uniformly. However, depending on the content of the resulting tree by comparison with existing parts of the representation, it is possible to retrospectively assign a category (e.g. by following the guidelines given in section 3.2.2 on page 36). For example, the tree inferred in cycle 21 is predictive, as it takes an existing *come* event and infers a *hurt* event, such that the *come* event precedes the *hurt* event; whereas the tree inferred in cycle 42 is explanatory, as the two instances connected (*kill giant* goal and *shooting* event) have already been observed at this point in comprehension. (This distinction between predictive and explanatory inferences is shown diagrammatically in figure 3.4 on page 40.)

Some comments on IDC's protocol:

- There is not a strict one-to-one correspondence between elements of the human protocol and IDC's protocol. The main reason for this is that some of the subtlety

of the human protocol is lost in translation, partly due to the need to add auxiliary inferences and partly due to my primitive modal notation.

- Some inferences which seem intuitively unavoidable and which are made in the human protocol are missing from IDC's protocol: for example, the inference that Ivan's waiting is part of an ambush. Some of these absences are due to the level of Skepticism set: IDC does have a schema for connecting sub-nodes into an *ambush* event, but this schema is too unreliable to be applied when Skepticism = 0.5. Similarly, the human protocol contains a predictive inference that the people celebrated because they were overjoyed; with Skepticism = 0.5, this inference is also missed. However, both inferences *are* made when Skepticism is set to 0.1.

The rest of the absences are due to problems with mapping between the human protocol and IDC schemas. For example, the human protocol includes the inference 'Ivan probably hopes that the giant is dead', which is very difficult to translate into IDC's restrictive notation.

- Some inferences are part of IDC's protocol but not part of the human protocol. These are generally inferences about causal relationships, e.g. cycle 32's action is inference of an enablement relation between the giant's coming to the village and Ivan's waiting in the village. In turn, this relation could be used to infer an *ambush* plan (see previous point), though this is not done when Skepticism = 0.5.

These inferences do not manifest in the human protocol because they are only implicit in the inferences reported. For example, the inference 'Ivan shot an arrow at him [the giant] and tried to kill him' does not explicitly mark the relation *Ivan's goal to kill the giant motivated his shooting the arrow*; However, causal relations are represented in Trabasso and Magliano's causal network for the story, which I used as another source for IDC's schemas (see figure 7.2).

Comparisons Between the Human Protocol and IDC's

Is there any meaningful correspondence between IDC's protocol and the human protocol, in terms of the time course of comprehension? By comparing the observations in IDC's text with the sentences in the English text, one can divide the propositions into groups, each of which corresponds to a sentence in the English text. For example, the first three statements in IDC's text can be mapped to the English text as follows:

habit(great_warrior, [exp:ivan]) = S1. 'Ivan was a great warrior.'

habit(best_archer, [exp:ivan, loc_in:village]) = S2. 'Ivan was the best archer'

habit(live, [exp:ivan, loc_in:village]) = S2. 'in his village.'

It is then possible to determine how IDC's inferences correspond to sentences in the text, and compare this with inferences in the human protocol, as shown in table 7.4 (next page). In the interests of clarity, the human clauses shown in the table are paraphrases of the actual clauses from the protocol; I have also ignored paraphrases in the human protocol and only included associations, explanations, and predictions (paraphrases are simply repetitions of information in the text and not generated as inferences).

The various problems described earlier in this section mean that there is not a close correspondence between the output produced by IDC and by the human comprehender. The central difficulty is in translating the ad-hoc protocol presented by Trabasso and Magliano into a machine-readable form. As a rough and generous estimate, IDC makes inferences which correspond with those of the human comprehender 40% of the time. For the other 60% of the time, IDC's inferences either have no counterpart in the human protocol, or have a counterpart which occurs earlier or later in the human protocol.

7.2.3 General Discussion

The monitoring and control of incoherence in IDC generates inference protocols which bear some resemblance to those of a human comprehender. The important feature of the human protocol is how inferences are made on-line to integrate the most-recently comprehended sentence with the evolving representation. Once sufficient coherence is established, comprehension continues. This pattern seems to correspond with the idea of establishing causal sufficiency suggested by van den Broek and colleagues (see section 5.2.4). The protocol seems to support their idea of sufficient causal explanation: an observation is processed until sufficient causal explanation has been established, then the next sentence is read. Predictive inferences are explained as follows: 'If the information in the prior text is highly sufficient for a consequence, then a specific inference is made.' [van den Broek et al., 1995]. In other words, if the text strongly suggests a particular continuation, that continuation may be inferred.

However, the idea of structural sufficiency embodied in IDC produces a protocol similar to the human one. In addition, IDC explicitly quantifies 'coherence need' in terms of

Sentence	Human Inferences	IDC Inferences
S1.	Ivan is good at fighting.	Ivan is a great warrior because he is a good fighter.
S2.	I have an image of a medieval archer.	Ivan is an archer.
	Ivan's fights with a bow and arrow.	Ivan uses armour, a bow and arrows, because he is an archer.
S3.		Arrows are a kind of weapon.
	Because Ivan is an archer he will try to kill the giant.	
	Ivan will try to kill the giant to stop the villagers being frightened.	
		The giant's terrorising the people constitutes an attack on the people.
S4.		The giant's terrorising the people psychologically causes Ivan to be angry.
	Ivan is getting angry.	
S5.		The giant's coming physically causes some hurting event, with the villagers as its patient.
	Because Ivan cares about the villagers, he wants to slay the giant.	
		The giant's terrorising the people psychologically causes Ivan's goal to kill him.
S6.		The giant's hurting the people psychologically causes Ivan's goal to kill him.
	Ivan is going to try not to be seen.	
	Ivan is trying to ambush the giant.	
S7.		The giant's coming to the village enables Ivan's waiting.
	Ivan tried to kill the giant by shooting an arrow at him.	Ivan's goal to kill the giant motivates his shooting the giant.
S8.	Ivan hopes that the giant is dead.	
		Shooting the giant physically causes hitting him.
		Ivan killed the giant by shooting him and hitting him.
S9.	The people were happy because the giant died.	
	The people were happy because they were no longer frightened.	
	The people celebrated because they were happy.	

Table 7.4: Comparison of human and IDC protocols

minimisation of incoherence, with respect to Tolerance and Skepticism, as follows:

1. The number and type of known representations which could explain/elaborate an r-elt (i.e. absolute incoherence);
2. How thoroughly the comprehender requires those representations to be explicated (i.e. Skepticism);
3. How much pressure is placed on the short-term store by r-elts currently maintained (i.e. Tolerance).

Of course, this does not prove that the human comprehender produced their protocol via the same or similar mechanisms (birds and planes both fly, but employ radically different mechanisms). I'm not claiming that human comprehenders explicitly employ incoherence criteria, merely that they are aware of the structural potential of the texts they are comprehending and attempt to make effective use of this potential. This differs from most previous models of coherence, where it is assumed that structure is added to the representation by inference; instead, I am claiming that structure is automatically inherent in the interaction between the text and a particular comprehension context. The comprehender's goal is not to establish structure, but to dismiss irrelevant structures and maintain relevant ones.

The distinction I'm making is subtle, but allows one to make sense (in the abstract) of ideas like 'causal sufficiency', 'standards for coherence' [van den Broek et al., 1995], 'requirements for understanding', and 'satisfaction of reader goals' [Graesser et al., 1994]. Without recognising their possible representations for a text, how can a comprehender decide when their representation is adequate? (see section 5.2.4). For example, how could a comprehender judge whether they had established sufficient causes for an eventuality, without knowing the range of causes available? As an analogy, consider quality control in a factory which produces 'widgets': without knowing the components which go together to make a widget, how could the quality controller know whether a widget was defective or not? Choosing one or more representations relies on a similar acknowledgement of the space of possibilities.

How would one demonstrate that human comprehension involves 'monitoring the possible representations space' rather than 'finding connections'? Superficially, as discussed above, the results/outputs of the two processes may be very similar, as incoherence and coherence are two sides of the same coin (see section 134). This means that distinguishing

incoherence-minimising from coherence-seeking may be difficult on the basis of outputs alone. Another problem is that IDC only models a fragment of the complete comprehension process: it is possible that comprehension of unusual texts relies on far more creative methods.

However, an interesting possibility suggested by IDC is that commonplace eventualities do not engender significant explanatory inferences, while the interaction between commonplace and less-common eventualities does.² This is because commonplace eventualities occur in many schemas, and so do not readily discriminate between them; IDC is thus reticent to actualise one of the potential explanations. But, in tandem with a more discriminating eventuality, a commonplace eventuality provides many opportunities for building structure into the representation. As a consequence, IDC tends to avoid making inferences on the basis of instances of the commonplace node, but quickly makes inferences where they occur in the company of discriminating instances.

As an example, consider the following texts:

Text 1

1a. John entered a building.

1b. He went to the counter.

Text 2

2a. John entered a bank.

2b. He went to the counter.

Given text 1, it is unlikely that a comprehender would make significant inferences from sentence **1a** *alone*. It is possible that some inferences would be generated after **1b**, but I'd expect their quantity and specificity to be lower than those produced by text 2. Text 2 should produce more inferences on the basis of the sentence **2a**, as the specificity of the building involved is much greater. Sentence **2b** should also be easily processed, as it can be readily incorporated into a representation based on a 'bank visit' schema. In addition, one would expect this sentence to be processed relatively more quickly (with respect to sentence **2a**) than sentence **1b** (with respect to sentence **1a**).³

²Here, I intend 'commonplace' to mean nodes which have many possible explanations, and thus high ubiquity (e.g. *go* events).

³Of course, Skepticism may allow generation of inferences on the basis of a commonplace eventuality alone. For example, with a low enough Skepticism, the statement 'John entered a building' may trigger a large number of explanatory inferences (see next section for more details).

Coherence theories cannot adequately account for comprehension time differences in the above types of circumstance. While these theories could be used to make similar predictions by invoking ideas of global coherence, availability of schemas and so forth, they have never satisfactorily defined why/whether general texts should produce fewer inferences than more specific ones; in particular, there are no computational theories in this vein. Many coherence theories based on referential and/or causal connectivity would fail to distinguish between the above texts, treating them as equally coherent. In each text, both sentences can be connected causally, for example by an *enables* relation (entering the building/bank enables John to go to the counter) (see section 7.2.1). They thus seem to be equally coherent in the sense of allowing an equal number of direct causal relations.

However, it is still possible that they are not equally coherent in other ways: for example, the strength of the causal relation may be greater in text 2, as the fact that John enters a bank, rather than an arbitrary type of building, increases the likelihood of his going to *some* counter. According to van den Broek's equation for causal strength, the greater the sufficiency of a cause for its effect, the greater the causal strength between them [van den Broek, 1990a]. This suggests that van den Broek is echoing the view that coherence is tied to probability, following the lead of work such as [Smolensky, 1986] and [Thagard, inpr]. A problem here is denoting what the probabilities represent: as I argued earlier, the only way in which probabilities can make sense is with respect to the knowledge base (see section 5.3 on page 94).

Coherence is often equated with how well-connected elements of a representation are, but texts 1 and 2 hint that connectivity must be determined with respect to a knowledge base. There has been little previous work on the processing load induced by 'number of alternative representations'. One notable exception is the work of Sanford and Garrod, some of whose work is similar to (and has influenced) my incoherence theory [Sanford and Garrod, 1981]. In their 1981 book, they examine anaphora in discourse, stating that each term which introduces a new entity into a discourse also 'opens up [...] a range of potential anaphors' (ibid.). Depending on the specificity of the term, the range of potential anaphors may be wide (e.g. 'vehicle') or narrow (e.g. 'tank'). Given some subsequent, coreferential information, the time taken to integrate this information varies with the number of potential anaphors introduced by the preceding term: for example, if the word 'tank' is followed by the word 'vehicle', integration occurs more quickly than when 'vehicle' is followed by 'tank'. The incoherence theory I have developed in this thesis provides a computational slant on this idea, focused at the level of integrating eventualities

into plan and goal structures rather than resolution of anaphors.

In the next section, I discuss another aspect of IDC's behaviour: maintenance of multiple representations and representation revision.

7.3 Accounting for 'Role-Shift' Texts

My work describes how a comprehender's interpretation may be changed in response to new information. Although my original ideas didn't mention belief revision, I eventually realised that this was my principle concern: deciding when to create a new interpretation, and as a consequence delete all or some of a previous representation, according to the information available at a given point in time.

A similar process occurs in sentence-level comprehension, when so-called *garden path* sentences cause globally-incorrect interpretations to be constructed based on incomplete information. A famous example:

The horse raced past the barn fell. [Crain and Steedman, 1985]

The syntactic processor may create a parse of the first part of this sentence, *The horse raced past the barn*, which treats *The horse* as the subject of the sentence, *raced* as the main verb, and *past the barn* as a prepositional phrase, with *the barn* as the sentence's object.⁴ However, on reaching the end of the sentence, the reader has to 'undo' their interpretation, now treating *fell* as the main verb of the sentence.

In some garden path sentences, pragmatic information (e.g. world knowledge) allows discrimination between competing interpretations. However, garden-path-like effects can also occur at this pragmatic level. While the sentences of a text may be locally unambiguous, a text as a whole may encourage local interpretations which later turn out to be incorrect. By analogy with the syntactic effect, I call these *pragmatic garden path texts*.

One example of pragmatic garden paths involves *role-shifts* [Sanford and Garrod, 1981]. In such stories, 'garden-path'-like effects are experienced by comprehenders: they make default inferences about the roles of the characters in the story, which later turn out to be incorrect and have to be retracted. One famous example (adapted from (ibid.)) is:

⁴Various explanations of how this occurs have been put forward, such as attachment preferences [Frazier and Fodor, 1978] and parser heuristics [Pereira, 1985].

John was on his way to school.
 He was worried about the maths lesson.
 Last week he lost control of the class.
 It was unfair of the maths teacher to leave him in charge.
 After all, teaching wasn't part of a janitor's duties.

In this story, sentences 1 and 2 encourage the explanation that John is a schoolchild, worried about a maths lesson.

Sentence 3 undoes the initial explanation: as John lost control of the class, he must have been in control of the class; normally, teachers are in control of a class, not schoolchildren; therefore, John must be a schoolteacher. His worry can then be explained retrospectively in terms of his fear of failure as a teacher.

Sentence 4 again upsets this explanation: John isn't a teacher, but someone unfairly left in charge of the class. The previous explanation, that John is a schoolchild, may be reactivated: schoolchildren may be left in charge of a class. The worry may again be re-interpreted, as worry about the responsibility of being in charge of a class, and losing control again.

Sentence 5 leaves some parts of the representation in place: John was still left in charge of the class, but is no longer a schoolchild - instead, he is definitely a janitor. By a stretch of the imagination, a janitor may be left in charge of a class, though this is unlikely.

This seems to intuitively reflect a normal comprehender's behaviour when presented with this text. In the next section, I compare IDC's behaviour with this intuitive description.

7.3.1 Revision of Representations

This section describes IDC's behaviour when presented with the 'Janitor Text' of the previous section and demonstrates how explanations are revised as comprehension proceeds. Each of the tables below shows results for a particular class of inferences. The numbers represent the processing cycle when an inference was first made, and the processing cycle when it was retracted in response to a new tree. Retractions occur when a new tree causes a previous one to become redundant or spurious, as discussed on page 149. The resulting behaviour is akin to syntactic garden-path effects at the level of plan/motivation recognition.

The results are shown for three different settings of Skepticism. In all cases, Tolerance

= 5.267 and Range = -1; on each run, IDC maintained a single representation throughout and had to revise this representation to account for new observations. The behaviour under these conditions differs from behaviour where multiple representations are maintained, as described in section 7.3.2.

<i>Inference</i>	Skept = 0.1	Skept = 0.5	Skept = 0.9
John is a schoolchild going to school to learn. Retracted	3 17	3 34	3 33
John is a teacher going to school to teach. Retracted	17 n/a	n/a n/a	n/a n/a
John is a janitor going to school to earn a living. Retracted	n/a n/a	34 n/a	33 n/a

Table 7.5: Inferences about John's reason for going to school

The inferences in table 7.5 are based around the observation of John on his way to school. IDC has three schemas which explain people going to school (others are obviously possible). Initially, there is equal evidence to support all three; the schoolchild explanation is selected (as it is the first schema matched, and the schemas are ordered so that the most used will be matched first). In the multiple representations condition, one or both of the alternatives may be explicitly represented (see next section).

Once IDC infers that John is a teacher (see next table), the inference that he is a schoolchild becomes unnecessary to explain him being on his way to school - the information that he is a teacher (supported by his worry about professional failure) is used in two separate trees, enhancing connectivity and lowering incoherence. This demonstrates how changes to trees can have a knock-on effect, causing the removal of seemingly unrelated trees.

However, note that in the Skepticism = 0.1 case, the inference that John is going to school to work as a janitor is not made, even though the text explicitly states that John is a janitor. Here, IDC's lack of a consistency checking mechanism is demonstrated, as John is assigned two roles, one as a teacher and one as a janitor. In the other two cases (Skepticism = 0.5, Skepticism = 0.9), no inference is made about John's status as a teacher, but the inference about him going to school to work as a janitor is made. The reasons for this are complex but chiefly due to maintenance of single, rather than multiple,

representations. This important point is discussed in more detail in the next section.

<i>Inference</i>	Skept = 0.1	Skept = 0.5	Skept = 0.9
John is afraid of a test during the lesson. Retracted	6 15	n/a n/a	n/a n/a
John is afraid of failing as a teacher. Retracted	15 29	n/a n/a	n/a n/a
John is a replacement teacher and is afraid of losing control of the class. Retracted	29 n/a	25 n/a	24 n/a

Table 7.6: Inferred explanations for John's worry

The interesting feature of table 7.6 is the relationship between Skepticism and the speed with which conclusions are reached. For example, the explanations that John is afraid of a test and that he is afraid of failing as a teacher are never generated when Skept = 0.5 or 0.9; however, the explanation that he is a replacement teacher who can't control the class is made earlier in these cases than in the Skepticism = 0.1 cases, mainly because incorrect inferences are not made which must be retracted.

Lower Skepticism causes IDC to 'jump to conclusions', which later may turn out to be incorrect; lower Skepticism also causes IDC to make more predictive inferences than high Skepticism. However, the higher Skepticism cases seem cognitively unrealistic as IDC comprehends the majority of the text before making any inferences at all.

<i>Inference</i>	Skept = 0.1	Skept = 0.5	Skept = 0.9
The maths lesson <i>m1</i> is a kind of lesson.	8	10	n/a
The maths lesson <i>m2</i> is a kind of lesson.	23	26	27
John is a janitor who has been drafted in as a replacement teacher.	33	35	32

Table 7.7: Auxiliary inferences which contribute to explanations

The final row of table 7.7 is the most interesting, as it shows how the Skepticism = 0.1 comprehender makes the connection between John being a janitor and being a replacement

teacher, while not retracting the inference that John is a teacher (see table 7.5). The latter inference is retracted in the other two cases, as the higher Skepticism comprehenders revise their explanation of John's going to school which results in removal of the instance corresponding to 'John is a teacher'.

The result of retaining this instance in the Skepticism = 0.1 case is that John is represented both as a janitor and a teacher. This intuitively seems like a contradictory state of affairs for a human comprehender. However, it may shed some light on inconsistency of representations in human comprehenders (see page 124).

<i>Inference</i>	Skept = 0.1	Skept = 0.5	Skept = 0.9
John is a child because he is a schoolchild.	9	4	4
The teacher t goes to school to teach.	n/a	24	22
The teacher t is an adult.	n/a	23	23
John attends the school because he is a janitor.	30	n/a	n/a
John is an adult.	16	n/a	n/a

Table 7.8: Irrelevant/elaborative inferences

Table 7.8 shows that some elaborative inferences (i.e. inferences which don't tie observations together) were made under all Skepticism conditions. A point of interest is how the higher Skepticism settings require elaboration of the representation of the teacher t , as *teacher* instances have several potential elaborations which are not specified in the text. In this case, the inferences are made to 'specify' the details of the instance. In the lower Skepticism setting, this is not required, as the potential elaborations are more readily discounted.

7.3.2 Multiple Representations

The results of the previous section show how IDC behaves when maintaining a single representation. One central feature of this behaviour is IDC's concentration on *localised inferences*. If a new piece of information requires extensive revision of the representation in both the STS and the LTS, IDC may not make all the necessary revisions. This is the case where Skepticism = 0.5: this setting causes IDC to miss the inference that John is afraid of professional failure as a teacher (see table 7.6 on page 186).

However, if Range is adjusted (e.g. to 1) to allow multiple representations to be maintained, this inference *is* made. The reason for this is that maintenance of multiple representations gives greater scope for the integration of new observations as there are more possible attachment points for them. The result is that the inference about John's fear of professional failure is made in cycle 14 (note that the inference that he is worried about a test is still not made) and retracted in cycle 24.

More accurately, the reason for this inference being made when Range = 1 and not made when Range = -1 is because of the limitations of *gradient search* [Rich and Knight, 1991]. In this kind of search, all possible extensions to the current problem state are generated, then the best of the new states selected as the new problem state (c.f. IDC's behaviour when Range = -1). The problem with this type of search is that it can settle on *local maxima*: states which are optimal locally (i.e. where there is no better state within one 'move') but which are not globally optimal.

As an alternative, Range can be set to zero or greater, making IDC engage in *beam search*. This means that it potentially maintains multiple states (representations) on each cycle (depending on their divergence, as described in section 6.4.2). When new information is observed, it can be added to each of the current representations; hopefully, it will then be an easy matter to find a connection between the new observation and one of the existing representations. (Of course, beam search is not infallible, and it is often the case that IDC misses 'correct' representations.)

Returning to the example above (making the 'fear of professional failure' inference), the progress of IDC's interpretation when Range = 1 proceeds as shown in figure 7.2 on page 189. It is important to note that the diagram only shows inferences about John's role (schoolchild, teacher, janitor) and the reasons for his worry about the lesson (fear of professional failure, fear of losing control again). The arrows show the progress of comprehension; where there are multiple arrows leaving a cycle, this indicates a point where multiple representations are either derived or maintained. Where multiple representations are shown for a cycle, their quality (as determined by the incoherence metric) decreases from left to right: the left-most representation has the lowest incoherence and the highest quality, while the right-most representation has the greatest incoherence and lowest quality.

As can be seen from the diagram, three representations are initially generated, each of which assigns a different role to John (in the Range = -1 case, only the representation where John is a schoolchild is maintained). When the next two observations are processed,

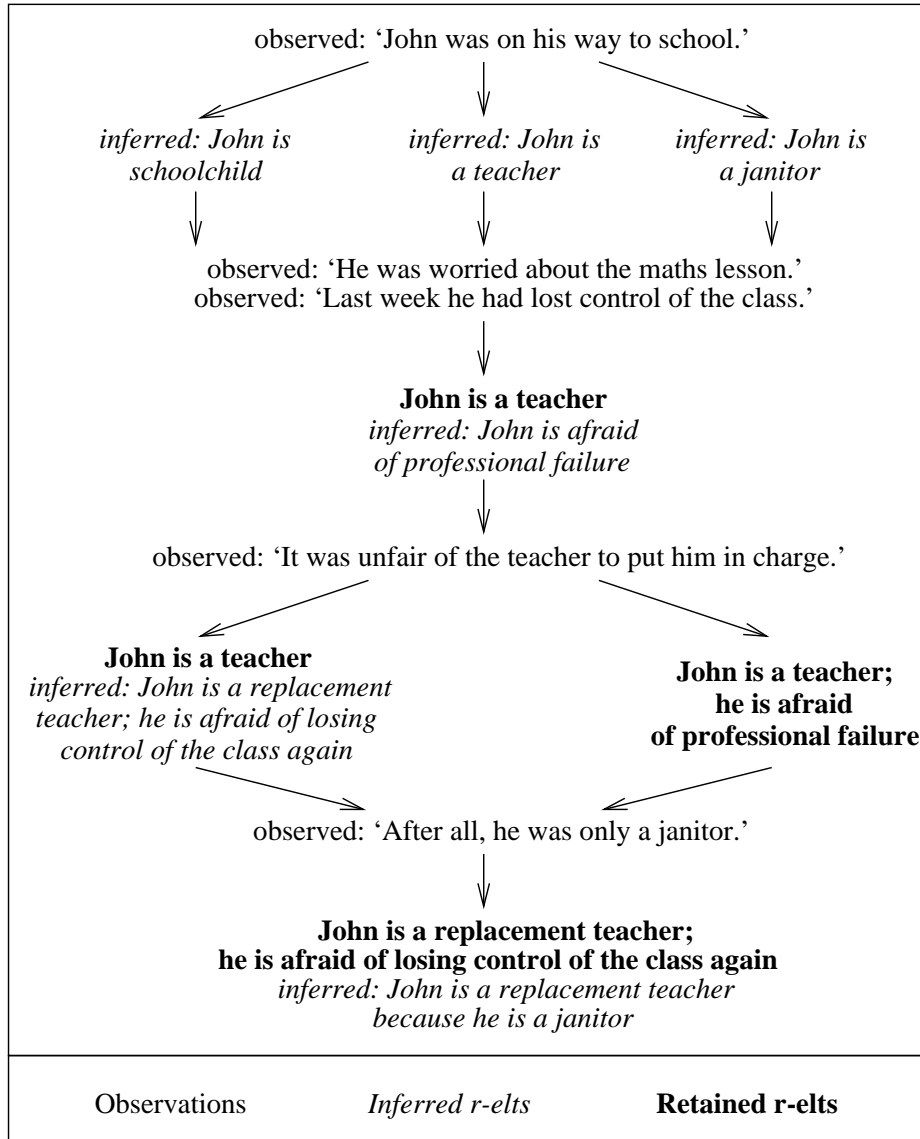


Figure 7.2: Multiple representations of the Janitor Story (Skepticism = 0.5, Range = 1)

it is a small step from the representation where John is a teacher to the representation where he is afraid of professional failure. However, an inference which increases incoherence is required to get from either of the other two representations to the 'professional failure' representation. As IDC inhibits inferences which create a positive change in incoherence (see section 6.4.1), it is unable to make either of these leaps. Range thus plays an important role in which representations are generated, especially in situations where there are several competing representations.

7.3.3 Implications for Human Comprehension?

In the two previous sections, I showed IDC's behaviour when maintaining a single vs. multiple representations. The important differences between the two cases are:

- The multiple representation case (i.e. Range is not equal to -1) requires greater storage capacity when a text prompts alternative representations (for example, the role-shift stories prompt multiple representations as they are ambiguous as to the role of the main character). Of course, storage capacity required is only greater if maintaining many elements in the STS requires greater storage capacity than maintaining fewer elements: hopefully, this is an uncontroversial point.

In general, this may cause the multiple representations comprehender to store alternatives for unambiguous texts as well as ambiguous ones, which may lead to inefficiencies and longer reading times. But, in cases where a text strongly suggests a single representation, the multiple representations comprehender may only maintain a single representation anyway.

- Maintenance of multiple representations provides more possibilities for attaching new observations to a representation (as there are more representations available). Opportunities for integration may be missed in the single representation case due to excessive distance from the current representation to a representation which can incorporate an observation (see the comments on gradient search in the previous section).

I have tested these predictions by comparing IDC's data for role-shift and non-role-shift text comprehension to human data. Human data for these texts concentrates on the reading time for sentences where the role-shift actually occurs [Sanford and Garrod, 1981]. For example, Sanford and Garrod asked readers to produce continuations of the sentence

'John was on his way to school'; they found that those readers who mentioned John's role 'indicated that he was being thought of as a schoolboy' (*ibid.*). They then used this information to produce two versions of a variant of the Janitor Text to ascertain the time advantage for non-role-shift texts over role-shift texts:

1. Non-role-shift (John is assumed to be a teacher throughout comprehension)

1a. John was worried about teaching maths.

1b. He was on his way to school.

1c. Last week he lost control of the class.

2. Role-shift (from ambiguous role in first sentence to 'teacher' role in final sentence)

2a. John was on his way to school.

2b. He was worried about the maths lesson.

2c. Last week he lost control of the class. [Sanford and Garrod, 1981]

Their hypothesis is that when role changes occur (e.g. in sentence **2c**), comprehension time for that sentence is increased (*ibid.*). This is due to the need for mental reorganisation, as the assignment of a schoolboy role to John is found to be incorrect and must be revised (e.g. by assigning John the role 'teacher'). They found that the reading time for sentence **2c** was 10% greater than that for **1c**, indicating that extra 'work' was required for revision of the representation and supporting their idea that representations are produced very rapidly, even for short texts.

The Time-Course of (Non-)Role-Shift Text Comprehension

What can IDC tell us about representation revision of the sort described above? One problem in answering this question is the difficulty of precisely aligning human reading times with IDC's comprehension cycles. Unlike models which limit the work done on each cycle (e.g. Just and Carpenter's CAPs model [Just and Carpenter, 1992]), IDC cannot be directly compared with measures such as gaze duration [Thibadeau et al., 1982] as its cycles are not limited in the amount of work they can do. However, it is possible to look at IDC's protocols on different texts, under different Range conditions (i.e. maintaining single or multiple representations), and use these to determine the amount of time that IDC spent comprehending the individual sentences of the two texts. Unlike Sanford and

Garrod's data, this data is based on the relative amounts of time spent on all of the sentences of a text; it thus suggests the kinds of results which may be expected if factors such as Range and Skepticism are manipulated in human comprehenders, as described in section 7.3.4; it does not predict the precise amount of time (e.g. in seconds) that comprehenders will spend on individual sentences.

I produced some IDC protocols and calculated the length of time IDC spent on comprehension of sentences in the two texts (role-shift and non-role-shift) as follows:

1. I ran IDC twice on each of the two texts, for a total of 8 runs, with the following parameters:
 - (a) Text = 1, Skepticism = 0.1, Range = -1 (single representation)
 - (b) Text = 1, Skepticism = 0.1, Range = 1 (multiple representations)
 - (c) Text = 1, Skepticism = 0.5, Range = -1 (single representation)
 - (d) Text = 1, Skepticism = 0.5, Range = 1 (multiple representations)
 - (e) Text = 2, Skepticism = 0.1, Range = -1 (single representation)
 - (f) Text = 2, Skepticism = 0.1, Range = 1 (multiple representations)
 - (g) Text = 2, Skepticism = 0.5, Range = -1 (single representation)
 - (h) Text = 2, Skepticism = 0.5, Range = 1 (multiple representations)

In all cases, Tolerance was set at 2.

2. For each run, IDC calculates a *load* for each cycle. Load is equal to the incoherence of the unique r-elts across all representations maintained on a cycle; it is thus a measure of the total storage cost for the interpretation, or the amount of working memory resources required to maintain the interpretation.

I used the total load for the whole run divided by the number of cycles in the run to give average load per cycle, A .

3. The time spent on each cycle (T) was calculated using A and the load for that cycle (L):

$$T = \frac{L}{A} \tag{7.1}$$

4. The cycles in each run were then split depending on the sentence to which they applied. The first step was to decide which observations in the input belonged to which sentence of the text, e.g. for text 1 (sentences **1a** to **1c**):

1a. = *event*(*e2*, *worry_about*, [*agt:john*, *pat:m1*]),
 habit(*teacher*, [*exp:john*]),
 habit(*math_lesson*, [*exp:m1*]).

1b. = *event*(*e1*, *go*, [*agt:john*, *loc_to:s*]),
 habit(*school*, [*exp:s*]),
 relation(*precedes*, [*e2*, *e1*]).

1c. = *event*(*e3*, *lost_control*, [*agt:john*, *pat:m2*]),
 habit(*math_lesson*, [*exp:m2*]),
 relation(*precedes*, [*e3*, *e1*]),
 relation(*precedes*, [*e3*, *e2*]).

For each run, I used these groupings to split the cycles into groups, with each group representing the set of cycles during which a single sentence was being comprehended. I did this by marking the output from each run: cycles occurring from the first observation of a sentence S_i to the first observation of sentence S_{i+1} were treated as relating to sentence S_i .

5. Given the cycles relating to each sentence, it is possible to calculate the total load experienced during comprehension of that sentence.

The result of these calculations is a list of three numbers for each run, describing the relative load during comprehension of each sentence of a text. For example, the loads for each sentence of text 1 when Skepticism = 0.1 and Range = 1 are:

1a. 4.708
1b. 8.905
1c. 9.387

Figures 7.3 and 7.4 (on the next page) show the relative times spent by IDC on each of the sentences of the non-role-shift (text 1) and role-shift (text 2) texts respectively; in all four cases, Skepticism = 0.5. In each diagram, the results for a single representation

(Range = -1) version of IDC and a multiple representations (Range = 1) version are shown: the single representation comprehender with Skepticism = 0.5 is denoted $C_s^{0.5}$ and the multiple representation comprehender with Skepticism = 0.5 $C_m^{0.5}$.

The graphs have some interesting features:

- In figure 7.3, $C_m^{0.5}$ spends more time on sentence **1a** than $C_s^{0.5}$. However, $C_m^{0.5}$ spends relatively less time on comprehending the next two sentences. This is because $C_m^{0.5}$'s comprehension of the first sentence makes several alternative representations available for when the second sentence is comprehended; the observations of those sentences are thus 'slotted' into whichever representation best accommodates them, and other representations may be discounted. Although there are no striking ambiguities, as there are in the role-shift text, there is still potential for alternative representations, e.g. alternative elaborations.
- In figure 7.4, the time spent on the second sentence by $C_m^{0.5}$ is greater than that of $C_s^{0.5}$. The reason for this is that $C_m^{0.5}$ maintains several explanations for why John might be worried about the maths lesson, e.g. he is a teacher worried about failing professionally or a schoolchild worried about a test. However, by the third sentence, the information that John lost control of the class in the previous week causes these multiple representations to be collapsed to a single representation, that John is a teacher afraid of failing professionally. $C_s^{0.5}$, meanwhile, ignores the new information and retains the 'John is a schoolboy' representation due to the myopia I described in section 7.3.2: it is unable to integrate the information that John is worried about the lesson, and as a result has to maintain isolated information, creating a higher overall load and longer reading time.

If Skepticism is lowered to 0.1, the results are slightly different but in some respects more striking. These are shown in figures 7.5 and 7.6; the single representation comprehender is denoted $C_s^{0.1}$ and the multiple representations comprehender $C_m^{0.1}$. Some comments on these graphs:

- $C_s^{0.1}$ does not suffer the myopia of $C_s^{0.5}$ (see previous bullet points): it is able to integrate the observation 'John lost control of the class in the previous week' as the lower Skepticism allows it to pass the local maximum. As a result, $C_m^{0.1}$ does not gain an advantage on this text and spends more time on comprehending it. This reflects the prediction made in section 7.3.3, that comprehenders who maintain

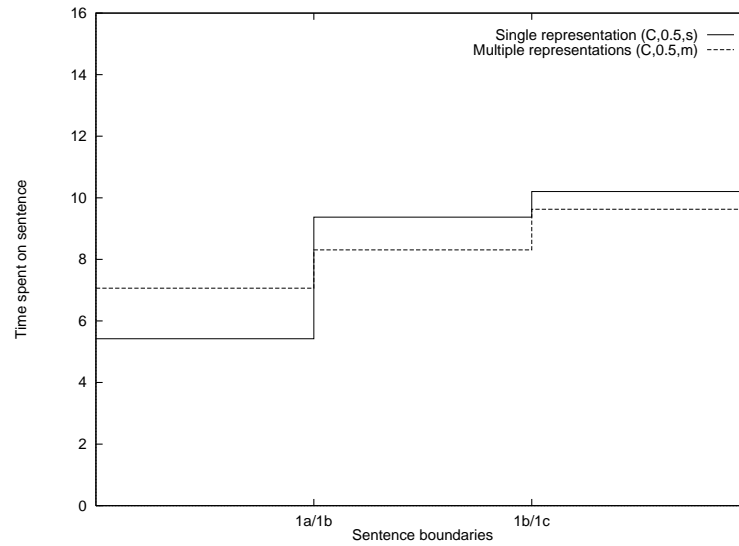


Figure 7.3: Relative time spent on sentences of the non-role-shift-text (text 1), Skepticism = 0.5

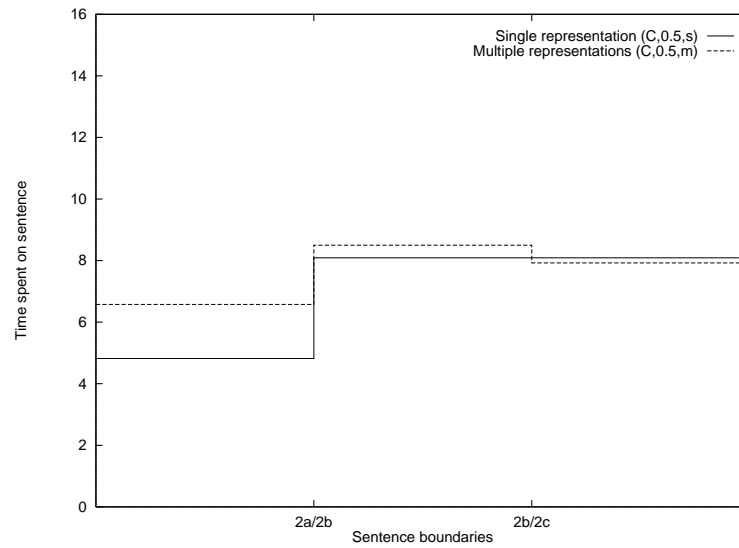


Figure 7.4: Relative time spent on sentences of the role-shift-text (text 2), Skepticism = 0.5

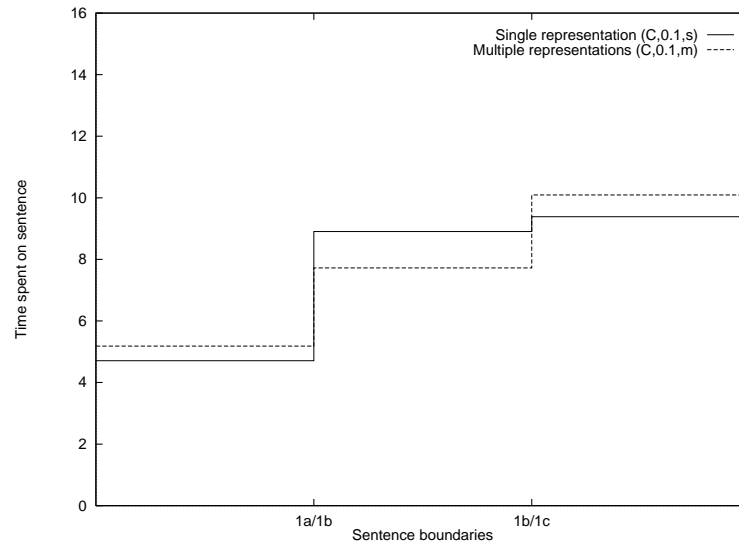


Figure 7.5: Relative time spent on sentences of the non-role-shift-text (text 1), Skepticism = 0.1

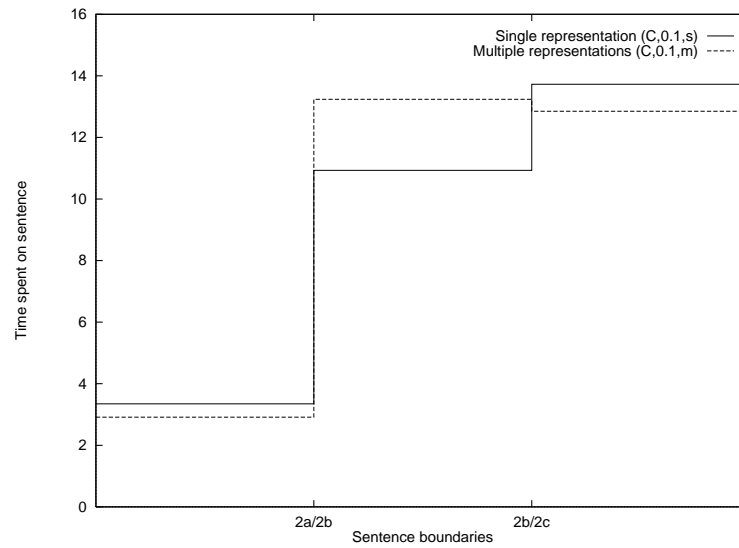


Figure 7.6: Relative time spent on sentences of the role-shift-text (text 2), Skepticism = 0.1

multiple representations are disadvantaged on unambiguous texts. This effect does not manifest itself when Skepticism = 0.5, as most potential representations are immediately discarded, being too incoherent for serious consideration.

- $C_s^{0.1}$ and $C_m^{0.1}$ spend more time on the role shift text than $C_s^{0.5}$ and $C_m^{0.5}$. This is because the lower Skepticism allows more unnecessary inferences to be made and requires more revision when new information becomes available. However, $C_m^{0.1}$ spends less time on sentence **2c** than $C_s^{0.1}$; again, the availability of multiple 'attachment points' (in various representations) results in more effort being expended on sentence **2b**.

7.3.4 General Discussion

These results demonstrate the complexity of comprehension and the myriad factors which influence the length of time taken to comprehend a given sentence. To summarise:

- Interactions between Skepticism and the number of representations maintained can complicate the reading time data, for example where maintenance of a single representation results in inferential myopia.
- If multiple representations are maintained, reading time for the third sentence of the role-shift text is *lower* than that for the second sentence. In the single representation case, reading time for the third sentence is *higher* than for the second sentence.
- At low Skepticism values, the reading times for role-shift texts are *higher* than those for non-role-shift texts, due to the need to revise representations. At higher Skepticism values, there is a tendency for reading times to 'level out', as there is less of a tendency to make early inferences which later require correction.

While these are only tentative conclusions, they present some interesting avenues for experiments involving human subjects. The important factors influencing reading time in IDC are Range and Skepticism, both of which could be manipulated in human comprehenders. I have so far defined both of these parameters as aspects of the central executive (see sections 6.4.3 and 5.3.6 respectively). Is there any reason to believe these parameters are separable in human comprehenders?

As I've stated earlier, *Range* monitors the quality of the whole interpretation: it determines whether alternatives to the least incoherent representation should remain as part

of the interpretation. *Skepticism* is responsible for monitoring the quality of individual representations: it determines whether individual inferences should be made within a given representation. In previous psychological accounts of strategic control of comprehension (though they are few in number), these ideas (interpretation monitoring vs. representation/inference monitoring) have rarely been considered together. The issue of multiple representations is not seen as important in accounts of interpretation revision: the emphasis has been on revision of a single representation, rather than switching between alternative ones (e.g. [Johnson and Seifert, 1999], [van Oostendorp and Bonebakker, 1999]). Where switching *is* considered (e.g. [Gernsbacher et al., 1990]), switches go from a current representation to a new one. At the same time, psychological accounts of inferential control rarely mention the possibility of inferences occurring at differential rates, dependent on a Skepticism-like parameter (with the exception of [van den Broek et al., 1995]). Instead, there is a focus on particular types of goals which may drive comprehension, such as goals engendered by reading literature [Graesser et al., 1994] or goals to maintain a certain class of coherence structure (e.g. superordinate goals [Long et al., 1996]).

As a consequence of the failure to consider strategic monitoring of coherence, both types of account are missing a description of its underlying 'abstract' mechanisms. These mechanisms can be thought of as the machinery which is employed while processing particular types of goal; for example, both 'literary' goals and 'news' comprehension goals may be implemented by running a single 'thresholding machine' [Zwaan, 1996].

In IDC, the Skepticism and Range parameters play the role of these abstract mechanisms. To determine whether they bear any resemblance to the underlying machinery in human comprehenders, it is necessary to speculate on what could affect one parameter while leaving the other intact; or, from another perspective, how one parameter could be manipulated without altering the other. This would then demonstrate that Skepticism and Range are separable, and thus add weight to my previous suggestions about how these factors affect human comprehension (i.e. how they control the generation of inferences and maintenance of alternative representations). It may also shed some light on various processing disorders, as described in section 8.3.2.

Because it is difficult to get at these abstract mechanisms directly, the only way to access them is by changing the comprehender's goals. Two techniques for manipulating Skepticism and Tolerance are given below:

- *Skepticism*: change the comprehender's goals by giving instructions to be more or less imaginative/fanciful in making inferences. As Skepticism represents the

comprehender's caution in jumping to conclusions, one possibility would be to change the reading situation between groups of subjects. For example, both groups could be presented with a text, with one group being told the text is a news story (high Skepticism) and another told the text is a piece of fiction (low Skepticism) [Zwaan, 1996].

- *Range*: change the comprehender's goals to encourage more or less alternatives to be maintained. One possibility would be to encourage fast, skim-comprehension in one group of subjects, and more leisurely, studied comprehension in the other. As maintenance of multiple representations requires more effort, the pressure to comprehend at a particular rate should influence the amount of time available for generating alternative representations; in the skim-comprehension condition, the pressure would hopefully force maintenance of a single, most-coherent representation.

An alternative piece of evidence which may strengthen the argument for a separate Range parameter can be found in research on children's recognition of alternative interpretations of texts. Bonitatibus and Beal found that older children were more likely to report multiple interpretations of a text than were younger children [Bonitatibus and Beal, 1996].⁵ However, in both the younger and older children, the interpretation formed is 'complete' in that it provides a reasonable account of the text's events. This suggests that older children have a greater Range than the younger children, while both could have equivalent Skepticism. Another interesting point is that younger children are less able to repair interpretations in the face of conflicting information: perhaps they are suffering from 'inferential myopia', caused by their inability to maintain multiple representations during comprehension (see section 7.3.3)?

Some evidence which undoes this reading of the results comes from a variant on the experiment, where children were informed that the text was either a narrative or an expository text (c.f. previous bullet point). In this case, it was found that both the younger and older children reported more alternative interpretations. However, in situations where subjects are asked to report on their understanding it is always difficult to avoid the possibility that the attempt to answer a question causes construction of an alternative interpretation 'on the fly', rather than uncovering

⁵Interpretation is used here in the common sense of explanation for a story's events.

a genuine multiplicity of representations. (Below, I suggest some tests for multiple representations which can be carried out *during* comprehension.)

The reading times in the various circumstances (produced by varying Range and Skepticism via goal manipulation, and text type) could be gathered using a technique such as self-paced reading, where subjects push a button after completing each sentence [Graesser et al., 1997]. However, this is a primitive technique at best. A technique such as gaze duration analysis is a more advanced possibility [Thibadeau et al., 1982]. With this technique, eye movements during comprehension are measured (usually for individual words); the hypothesis is that the length of time spent processing individual words reflects the cognitive effort spent on processing those words (the 'eye-mind' hypothesis (*ibid.*)). Individual eye movements for words which trigger role-assignment in the comprehender (such as 'school', 'teaching') could be measured to determine the amount of processing carried out on reading these words. (However, it would be necessary to factor out effects such as word length, word frequency, etc. [Graesser et al., 1997].) These reading times could then be compared with IDC's to determine whether the effects mentioned at the start of this section occur in human comprehenders. For example:

- Does the word 'school' engender more processing in the multiple representations condition than the single representation condition, because multiple roles may be assigned to John?
- Does the word 'worried' engender more processing in the role-shift text (where John's role is ambiguous) than in the non-role-shift text (where it can only really be interpreted in the context of John's concern about professional failure)?
- In the multiple representations condition, are reading times for the third sentence of the role-shift text shorter than reading times for the second sentence?

IDC's reading times suggest positive answers to all of these questions.

A final point: it is perhaps controversial to claim that comprehenders maintain multiple representations *at all*. To support the above experiments, it would be necessary to determine whether the multiple representations maintained by IDC are in fact mirrored in human comprehenders; for example, by testing for activation of roles during comprehension of role-shift and non-role-shift texts. Otherwise, any similarities between human reading times and IDC's may be attributable to other effects, such as lexical access, word familiarity, etc.

As I showed in section 7.2.2, IDC can be used to give some clues about the time-course of concept activation. This information could be used alongside expert judgements to determine where roles (and other concepts) are likely to be active during comprehension of the role-shift and non-role-shift texts. The resulting list of potential concept activations, tied to sentences of a text, could be employed to carry out a *lexical decision task* [Long et al., 1990].⁶ In this task, a subject is presented with a letter string during comprehension and required to decide whether the word is a word or a nonword (ibid.). If a concept is active in memory, judgements about words related to that concept should be faster than judgements about words not attached to the concept.

To test whether multiple representations are generated during comprehension of the role-shift (ambiguous) text, the first step would be to derive a list of letter strings relating to the roles in the text (e.g. 'teacher', 'schoolchild', 'janitor'). A comprehender would then be asked to read the text. After reading a particular sentence, they could be probed with a letter string from the list, such as 'teacher'. The time to respond (i.e. to make a decision about whether the word is a non-word or a word) is then recorded. These results would have to be compared to those for the non-role-shift text, to ensure that the *text* is causing multiple representations to be generated, rather any other factor (such as the comprehender's world knowledge). For example, concepts hypothesised as being activated in the role-shift condition but *not* in the non-role-shift condition could be probed for during non-role-shift comprehension.

If the decision time for a probe relating to a role is faster than the decision time for unrelated letter strings and non-words, this is taken as evidence that the role is encoded into memory at the point when the probe is administered.

The results of these experiments could then give some idea of whether multiple representations are indeed maintained by comprehenders who are confronted by ambiguous texts. It would also be necessary to test for multiple representations under low and high Range conditions, by manipulating the comprehender's interpretation monitoring mechanisms (see page 198). Under the low Range condition, IDC predicts that only a single role is likely to be activated, regardless of the ambiguity of the text; there should thus only be a decision time advantage for the letter string relating to this role.

It is also important to point out that the explicit multiple representations maintained by IDC may not be reflected in the comprehender's interpretation. For example, if the representation merely consists of a set of activations across nodes (as in the Construction-

⁶This idea was suggested to me by Mark Torrance.

Integration Model), the results would be the same, without the interpretation being explicitly divided across representations. The contrast is between the *partitioned relations strategy* simulated by IDC (see figure 3.7 on page 48) and the implicit representation of alternatives in a symbolic-connectionist model (see figure 3.11 on page 57). It is difficult to say how one would test which of the possible ways of maintaining multiple representations is actually implemented in human comprehenders; the advantage of making maintenance explicit, as in IDC, is that the instances belonging to each representations can be separated out from irrelevant ones and reported to the outside world.

7.4 Chapter Summary

In this chapter, I showed how I have experimented with IDC on various types of text comprehension problem, from pure AI (plan recognition) to psychological (time course of comprehension):

- *Plan Recognition*

At the very least, IDC is a reasonable plan recognition program, even though my emphasis on hesitations and mistakes in comprehension makes quantitative judgements difficult. This emphasis is the very feature of IDC which also distinguishes it from previous research: although some researchers have designed quality metrics, there have been few who have approached the time course of plan recognition in this context. IDC demonstrates how decisions about whether to make inferences or not ('weighing of evidence') can be founded on the notion of incoherence.

- *Inference Protocols*

IDC shows how inference protocols can be produced using the same incoherence metric as is used for plan recognition. I feel that this work has been least successful, as it is difficult to tie the output of the program directly to a human protocol. The reasons behind this are manifold (see section 7.2), including factors such as:

- Difficulty in translation from a human protocol to a computable target representation (i.e. something for IDC to aim at).
- Restrictions of the ontology in IDC (e.g. lack of support for modal statements, primitive temporal representation).

- Problems with uncovering the ‘micro-inferences’ which would describe the human protocol at a fine level of detail.
- Unpredictable interactions between IDC’s parameters which cause inferences to be generated in the incorrect order or be missed (see section 8.5).

- *Role-Shift Comprehension*

Some of my most interesting results come from my work on role-shift comprehension. I have used IDC as the basis for some predictions about text comprehension:

- The kinds of processing likely to cause difficulties for the comprehender (e.g. maintaining single vs. multiple representations, low vs. high Skepticism).
- Some experiments which could shed light on whether comprehenders maintain single or multiple representations.
- How some comprehenders may suffer ‘inferential myopia’, where correct inferences are missed because of an inability to maintain ‘bridging’ representations (bridging representations allow local maxima to be avoided).

In the next chapter, I discuss the more general achievements, problems and conclusions resulting from my work.

Chapter 8

Conclusion

Early chapters described the theoretical basis of the incoherence model and its computational implementation in IDC. The previous chapter demonstrated how the implementation could be applied to various issues in text comprehension. In this chapter, I provide an overview of the work as a whole, emphasising my achievements and the difficulties raised by the research.

8.1 Evaluation and Summary of Research

IDC represents one step towards a complete model of text comprehension. It covers only a fragment of the work required to produce a complete model, something which has not been satisfactorily achieved by any researcher in this field. Obvious shortcomings of my work are its limitations as regards extent of the comprehension processes: for example, there is no treatment of morphology, syntax, or semantics (in the normal sense of the term). These are issues which I have deliberately avoided (though, in the first few months of the thesis, I was convinced I could tackle all of these and more). Expenditure of more time on these areas would have resulted in a broad but shallow model, perhaps capable of superficially understanding stories from English to ‘representation’; however, there are *dozens* of programs with this kind of capability (e.g. [Lehnert et al., 1983], [Ram, 1991], [Williams, 1992]).

Instead, I chose to carry out a deeper study of a single, relatively muddled and under-explored area of comprehension: *strategic control*. Many previous psychological models avoid this topic, instead concentrating on the relatively automatic, associative-network style of processing [Kintsch, 1998]. Admittedly, these models work from individual words,

rather than an artificial, computer-style language as IDC does; in this respect, they are more ‘complete’ than IDC. On the other hand, these models avoid discussion of strategic inference.

As a way into the problem, I analysed comprehension systems (both psychological models and implemented models) in terms of artificial intelligence architectures. This resulted in a partitioning of the problem into several issues:

1. What kinds of semantic representation support comprehension?
2. What kinds of inference process are supported by those semantic representations?
3. What kinds of episodic representation are generated by those inferences?
4. How does a comprehender make decisions about which representations constitute their final interpretation?
5. Are there any ways in which the results of these analyses could be slotted into a computational model?
6. Can the output of such a model be compared with human behaviour in a meaningful way?

I feel that the work presented in this thesis goes some way towards clarifying issues 1 to 4: few other researchers have even recognised that these are issues. Issue 4 has been particularly neglected. Part of the reason for this oversight is the tendency to rely on partly ‘black box’ mechanisms, such as relaxation and spreading activation in symbolic-connectionist networks. Where strategic control is discussed, such as in [van den Broek et al., 1995] and [Graesser et al., 1994], there is often a difficult gap to bridge between the idea of strategic control and symbolic-connectionism. Work such as [Cooper and Shallice, 2000] and [Zwaan, 1996], along with implementations such as WanderECHO [Hoadley et al., 1994], go some way towards addressing these problems, but there is still much to do. Perhaps it is simply the case that connectionism does not provide the right level of description for control processes [Eysenck and Keane, 1995].

I feel I have made less headway on issues 5 and 6. Even though IDC does ‘work’, the results are difficult to analyse and there are fundamental problems in the implementation. As these are the main failings in my work, I describe them in more detail in the next section.

8.2 Evaluation of IDC

The theory of incoherence seems to me (intuitively) to be along the right lines. It corresponds with my personal experience of text comprehension and theoretically encapsulates coherence, while avoiding some problems such as how coherence and other quality metrics can be integrated. It is at the same time a computational philosophy of comprehension which has been influenced by recent work in literary theory (e.g. [Ronen, 1994], [Ryan, 1985]). Theoretically, at least, it provides a framework for analysis of several related, ‘structure-building’ tasks.

Simultaneously, there is a chasm between my theory and its computational implementation. As is often the case, it is difficult to create a program which lives up to one’s expectations. Some major problems in the implementation are:

1. Limitations of relations. Because temporal and causal information is treated in the same manner as categorisation information, certain types of reasoning are impossible due to IDC’s syntax for representing knowledge.
2. Lack of consistency checking.
3. Problems with comprehension management.
4. Problems with the metric.

Each of these areas is described below.

8.2.1 Limitations of Relations

A problem with treating causal relations as schemas is the impossibility of recursive definitions. IDC’s comprehension process is akin to categorisation, and so does not admit recursive structures: the idea of ubiquity and informativity rests on this. However, in a general reasoning system, relations of a certain type should be inferable on the basis of other relationships of the same type. For example, a common form of reasoning utilises the transitivity of temporal relations such as ‘before’, ‘after’, ‘during’, etc. [Allen, 1984]. This requires rules of the following form:

$$\textit{before}(A, B), \textit{before}(B, C) \longrightarrow \textit{before}(A, C).$$

In IDC, a schema such as this would break the informativity and ubiquity calculations: to determine these values for $before(A, B)$ and $before(B, C)$, IDC would have to determine them for $before(A, C)$ (and vice versa). In one sense, this is because the rules make no sense in terms of a paronymy (the basis of IDC's schema lattice); if the content of the rule is changed, this becomes clearer:

$$sheep(A), sheep(B) \longrightarrow sheep(C).$$

In other words, in IDC's terms the rule defines one category (*before*) in terms of itself, which is as ridiculous conceptually as one sheep being composed of two other sheep. The temporal reasoning rule is not partonomic and so cannot be handled sensibly by IDC.

Another problem with relations is the vague, ad hoc way in which they are handled. Schemas simply define causality in terms of the co-occurrence of two eventualities and a pre-defined temporal relation between them. The temporal relation is not derived by IDC but supplied in the texts it analyses. A complete system for causal and temporal reasoning would need to derive temporal relations on the basis of verb tense and aspect, combined with world knowledge (c.f. [Kamp and Reyle, 1993], [Moens and Steedman, 1988]).

These are major failings of the system; they are also inexcusable in some respects, as causal and temporal reasoning are central to comprehension [Trabasso et al., 1995]. However, putting them right would require a research project to produce a complete ontology for both partonomic and temporal reasoning (e.g. something in the style of the CYC system, which has been in development for 16 years and currently uses 1,000,000 rules [Cycorp, 2000]).

8.2.2 Lack of Consistency Checking

If IDC's Skepticism is set low enough, there is no mechanism which prevents inference of multiple, possibly competing explanations. Incoherence puts the 'brake' on generation of inferences: so long as an inference lowers incoherence change in the representation as a whole, there is nothing to prevent that inference being made (assuming that the schema's constraints have been met, as described in section 6.1.3 on page 138). As a result, IDC will occasionally produce conflicting explanations: for example, a story where John buys a milkshake in a diner then points a gun at the owner can cause confusion, with both *robbing* and *restaurant_visit* plans being inferred.

This problem seems to stem from the lack of a consistency checking mechanism, such as the one in ACCEL [Ng, 1992]. Such a mechanism would provide a list of 'NOGOODS'

after the fashion of the ATMS (see page 48). Each NOGOOD would specify a set of instances which are inconsistent and must not co-occur in a representation; any extension to a representations which included a NOGOOD would immediately be discarded.

While this sounds fine in principle, it is hard to define a list of NOGOODS which is broad enough to allow multiple explanations or elaborations whilst simultaneously excluding inconsistencies. In the example I gave above, is John actually dining in the restaurant, or robbing it, or doing both? How many of the subevents associated with restaurant dining would John have to be involved in for *restaurant_dining* to become a valid inference? On some occasions, someone may enter a diner with the intention of having a meal then robbing the diner.

Some possible alternative solutions to this problem include:

- The NOGOODS approach, where *either restaurant_dining* or *robbing* may be inferred, but not both. The choice between the two would have to be based on a criterion like simplicity (i.e. maintain the simplest explanation and assume the other one is incorrect).
- The ‘differential activation’ approach. If using a spreading-activation algorithm, the alternatives are maintained if neither ‘falls out’ of the representation during network relaxation. If one explanation is sufficiently more activated than another it is maintained while the other becomes deactivated.

In the symbolic-connectionist paradigm, there is the suggestion that the best interpretations will result from such processes. However, as I showed in section 4.3.3 (page 77), this is sometimes because legitimate competitors have been pruned from the competition, by the action of an unseen and undocumented strategic inference processor: the researcher.

- The ‘compound schema’ approach. A special schema describing robberies involving decoys could be added to the knowledge base. This would allow both John’s buying the milkshake and pointing the gun to be seen as part of a *robbery_with_decoy* plan. This partially works in IDC, as a tree based on the compound schema will have less incoherence than two trees based on the *restaurant_dining* and *robbing* schemas.

The problem here is the combinatorial explosion of the knowledge base. For every likely combination of schemas, one would require a compound schema (visits to the restaurant which are also birthday parties, trips to the beach which are also marriage ceremonies, dinner parties which are also business meetings).

- A mixture of the above strategies. However, even if one had combined schemas, it might be the case that prohibition of representations based on a certain combined schema and a certain simple schema would be required. For example, it would be necessary to ensure that the *robbery_with_decoy* schema and the *restaurant_dining* schema are never used to create two trees in a single representation. The need for prohibition may be lessened for models which employ representation quality metrics such as coherence or simplicity.

This issue is a perennial problem in comprehension systems and not one I have solved. I have attempted implementations of several of these methods but none have proven satisfactory.

My own feeling is that explicit negation (NOGOODS) and compound schemas are unworkable; differential activation seems to be the best alternative (to a certain extent, this is the strategy IDC employs anyway).

8.2.3 Problems with Comprehension Management

IDC's comprehension management consists of the 'meta-control' processes which guide construction of the interpretation. Problems with these processes originate primarily from technical aspects of the implementation, such as how existing trees are replaced by new ones.

Some comments on technical problems with IDC:

1. *Failure to maintain reasonable alternative representations*

IDC's interpretation is composed of one or more alternative representations. If the incoherence of the best representation is above Tolerance, IDC attempts to generate a tree in each representation which lowers the incoherence of that representation. The problem here is that representations tend to converge on a single, best representation. If a new tree is generated in a representation, older trees may be replaced where they become unnecessary (see page 149). So, for example, a hypothetical tree T_1 , initially maintained as an alternative to another tree T_2 and explaining the same observations but with higher incoherence than T_2 , may be replaced by a newly-inferred 'replica' of T_2 . This is not to say that there will be multiple identical representations: IDC ensures that duplicates are removed from the interpretation. However, the diversity of the representations tends to disappear as alternative trees are displaced by the trees for which they provide alternatives.

2. *Inefficient use of search*

When IDC constructs extensions to a representation R , it finds all possible extensions which lower the incoherence of R ; these extensions are then sorted and trimmed, as described in section 6.4.2 (page 151). However, this is a source of inefficiency: *all* possible extensions are generated before sorting and trimming begin. A better policy would be to halt generation of extensions on the basis of available resources, following the lead of Capacity Constrained Comprehension [Just and Carpenter, 1992] or WanderECHO [Hoadley et al., 1994]. Using such a control would prevent construction of extensions once the capacity for the cycle was ‘consumed’. While this would in effect act as another kind of ‘beam search’, the point where control is applied is different, corresponding (approximately) to step 4 of figure 2.2 (page 21).

A heuristic based on schema size could be combined with this idea: matches against the knowledge base could be ranked on the basis of schema size, with smaller schemas being preferred. Then, schema applications could be applied in order of schema size, smallest first, up to the capacity limits suggested in the previous paragraph. This corresponds to step 3 of figure 2.2 (page 21).

A solution to both problems would be better management of alternative representations. One possibility would be to allow *either* multiple representations *or* deletion of unnecessary trees. It is clear that these processes are related: deletion of unnecessary trees in a single representation equates with retracting alternatives in the multiple representations case. However, in the current version of IDC, these processes co-exist. I again feel that a symbolic-connectionist approach may provide answers to this problem: multiple representations correspond to distribution of activation over nodes representing alternative ‘trees’, while deletion of unnecessary trees corresponds to deactivation of those nodes (see section 3.2.5, page 46).

8.2.4 Problems with the Metric

Perhaps the main problems with IDC involve its incoherence metric. As the metric is the core of my work, these problems are also the most serious.

One particular problem is the general arbitrariness of the metric. I had originally intended the calculation of informativity and ubiquity to be based on a probabilistic measure such as entropy, but still derived from the structure of the knowledge base (e.g.

following the work in [Bar-Hillel and Carnap, 1964]). The idea was to determine the utility of applying a particular schema in terms of how it reduced the entropy of the resulting representation; entropy would be related to the number of alternative possible extensions to the representation (*ibid.*). However, this approach depends on very unwieldy calculations and requires a definition of what the probabilities actually mean (see section 5.1.5 on page 91).

I then turned to logical methods for determining the utility of applying a schema, following the work of Appelt on model preference default reasoning [Appelt, 1990]. This formalism provides the semantics behind cost-based abduction.¹ However, the problem here was specifying how the comprehender's subjective control mechanisms (which eventually became Skepticism, Tolerance and Range) could influence costs on abductive rules.

The solution on which I eventually settled has little resemblance to any of the previous comprehension metrics. Incoherence is largely heuristic and incorporates aspects of coherence, similarity, competition and other metrics in a single measure. As a result of inventing these concepts virtually from scratch, some of the terminology associated with the metric is unfortunate, partly for historical reasons and partly due to the lack of a suitable vocabulary. The philosophy behind the metric also means that the comprehension process in IDC produces representations which are logically inconsistent, invalid and redundant: because I was interested in how a 'lazy' or 'poor' comprehender could be realised computationally, the system has loopholes which can undermine 'normal' comprehension.

IDC really needs to be placed on firmer ground to make it computationally sound. I think the most fruitful direction would be to find a probabilistic rationale for the idea of incoherence. I feel the idea of 'possible representations' is helpful in this light, tied to calculation of entropy with respect to a knowledge base. Some work which has already been carried out in this vein includes [Goldszmidt and Pearl, 1996] and [Nagao, 1993].

These improvements also need to be tied to a learning mechanism. At present, the hierarchy of knowledge in IDC is hand-coded; this makes its behaviour unstable and unpredictable as changes to schemas alter the representations which are derivable and their quality ratings. The effect is *nonmonotonicity in IDC's behaviour*: as the rulebase changes, representations which were previously seen as high quality may be discarded, while previously poor representations become more acceptable.

To remove these influences from the incoherence metric and make the schemas more

¹I have not explicitly discussed cost-based abduction, primarily because it has been shown to be a variant of probabilistic abduction [Charniak and Shimony, 1990].

‘objective’, one could write a program which derives the knowledge base automatically, e.g. from some kind of corpus. This would of course require a suitable corpus; as IDC’s knowledge base is used to make inferences about ‘meaning’, the corpus would also need to contain this information. One possibility would be to use a corpus of inference protocols and their associated texts: IDC could derive its rulebase by looking at the frequency with which a particular inference and a particular text segment co-occur; the resulting schemas could then be annotated with these frequencies, which could be used in the entropy calculations mentioned earlier.

There is little current research in this area. The main area of current corpus analysis research in text comprehension concerns the strength of associations between concepts, based on Latent Semantic Analysis [Kintsch, 1998]. This technique is being used to derive associative networks (used by researchers such as Kintsch) which can then be used in symbolic-connectionist modelling.

8.3 Extensions

I have hinted at some of the more mundane extensions to the model in previous sections: for example, adding other levels of processing (e.g. syntactic), improving the temporal ontology, and adding facilities such as question answering. However, I think these additions are not particularly interesting.

Instead, I think any future work would first have to deal with the metric. There is still much to do here and my formulation is by no means definitive. I describe some ways to improve the metric in the next section.

A second area where improvements need to be made is in aligning IDC’s protocols with human data. Potentially, analysis of comprehenders suffering executive function deficits could clarify the separation of various functions (e.g. whether Skepticism and Tolerance are really two separate parameters, whether Range is psychologically instantiated). This topic is covered in section 8.3.2.

8.3.1 Increasing the Formality of the Metric

As I hinted above, it would be interesting to overhaul the whole idea of incoherence using more formal techniques such as probability and connectionist relaxation algorithms. For example, the idea of potential representations obviously resembles possible worlds theory [Nagao, 1993] and model preference default theories [Appelt, 1990].

A possible technique for revamping the metric would be specifying the probability of a representation in terms of the probabilities in the space of representations. The probability that a particular representation is true could be based on frequency of occurrence of the rules used to construct it (with respect to the knowledge base); this is similar to my approach, but would require a more formal definition of frequency of occurrence. For example, this could be based on something as simple as IDC's 'number of times an element occurs as a consequent or antecedent'; or on something more sophisticated, such as the *firmness* of rules in [Goldszmidt and Pearl, 1996]. The latter work ranks each representation (worlds) on the basis of how many rules *tolerate* it.

Another approach could use actual protocols to determine the probability that a particular representation is correct. I envisage a procedure similar to case-based parsing here, although this is only a tentative suggestion.

8.3.2 Insights from Analysis of Executive Function

Skepticism and Range are two chief modulators of comprehension performance. As I've already suggested, one can examine how and if similar parameters influence human comprehension by examining protocols gathered in varying circumstances.

Another approach may be to attempt to gather protocols from individuals with assumed executive dysfunctions. In particular, individuals on the autistic spectrum and those with frontal lobe disorders have both been suggested as suffering from executive dysfunction. Thus, it may be fruitful to use these groups as cases for comparison with IDC. If their protocols are comparable to IDC's, this may give some support to the idea of 'using potential structure'. A corollary of this idea exists in the literature on autism in the form of the *weak drive for central coherence*.

In the normal cognitive system there is a built-in propensity to form coherence over as wide a range of contexts as possible. It is this drive that results in grand systems of thought, and ultimately in the world's great religions. It is this capacity for coherence that is diminished in autistic children. [Frith, 1989]

Frith seems to suggest that there is no drive in autistic individuals towards utilisation of structure. It is not that autism impairs semantic knowledge; some autistic individuals can have highly developed semantic knowledge bases while lacking the 'control mechanisms' which enable others to apply this knowledge:

Some individuals with autism possess a large store of information, but seem to have trouble applying or using this knowledge meaningfully. [Ozonoff et al., 1991]

This idea is comparable to aspects of my own model, where a rich knowledge base may be thwarted by lack of a drive to integrate information, resulting in impoverished representations. IDC's equivalent of central executive disorder is formed by increasing the level of Skepticism so that schemas have to be thoroughly instantiated by a text before an inference is made.

IDC may form the basis of a modularisation of the deficits which occur as a result of central executive dysfunction. For example, is the weak drive for central coherence really a single 'modulator', or are there component processes which together produce this effect? Is the weak drive the result of a failure to admit inferences (due to an influence like Skepticism), or the result of a phenomenon like inferential myopia (see section 7.3.3), caused by an inability to avoid local maxima? In addition to the model perhaps providing a framework for analysing executive dysfunction, I think analysis of protocols from such comprehenders could clarify issues in the model.

The main barrier to an analysis such as the one I've suggested is the failure to communicate intelligibly which often accompanies central executive dysfunction. I have not thoroughly examined ways to bypass this problem.

8.4 Achievements

From the beginning of my Ph.D. work I have been interested in the mechanisms which allow comprehenders to make sense of texts. My focus has always been on the ways in which representations can be compared with each other: this seems to me to be the central problem. The complexity of this task is increased when one realises that it occurs incrementally, based on a fragment of the comprehender's complete representation, and in the context of a theoretically unbounded inference process.

An early realisation was that metrics could be divided into two camps: those based on *explicit* annotation of rules, specifying an ordering on the desirability of their employment; and *implicit* measurement of the 'structure' and/or 'quality' of representations. My aim was to try to find a way to bring these two approaches together.

The novel parts of my thesis can be summarised as follows:

- I have unravelled some of the tangle of previous work, where considerations of metrics, representations (semantic and episodic) and processes are entangled. This gives a clearer idea of where effort needs to be directed by future researchers.
- I have designed and implemented a theory of ‘potential structure’. This shows one possible method for annotating a knowledge base without reference to the outside world. There are issues to resolve here: for example, where do schemas come from in the first place? Why are schemas organised the way they are?
- I have not sidestepped the (philosophical) issue of why comprehension occurs. My model uses ideas influenced by information theory to specify construction of representations in terms of ‘reduction of uncertainty’.
- I have incorporated some psychological data on comprehension into a computational model and suggested methods for comparing the model with human comprehenders.

A main result of my work is the finding that reliance on the structure of the knowledge base for rating representations makes a system very fragile and unpredictable. Ng claims that his system (ACCEL) is resistant to changes to the knowledge base, but this is only so if the rules are written in a particular way in the first place. The same is true of many symbolic-connectionist systems: they only reach the correct conclusions if the networks are designed correctly.

This is perhaps a slightly negative result, indicating that representation quality metrics based on the structure of the representation are too arbitrary; once you start looking at their underpinnings they seem to fall apart.

8.5 Last Words

I think a major criticism of my model may be its separation from ‘the real world’. As representations are accepted/rejected on the basis of other representations (not with reference to absolute probabilities outside the knowledge base), I may seem to be suggesting that comprehension is hermetically-sealed. In one sense I am: comprehension is an entirely subjective activity, governed by the vicissitudes of individual bias, error, ignorance, and imagination. However, commonalities of experience arise because people develop in a shared environment; it is the overlap in their experiences which gives rise to similar schemas, and hence some crossover in their interpretations.

My model suffers from the complexity of the interactions between its parameters: there are far too many of them, and slight adjustments to any one of them can cause havoc. However, this perhaps demonstrates that ‘normal’ comprehension exists along a relatively slim boundary between complete refusal to form representations and wild abandon. If nothing else, IDC gives an idea of the fragility and complexity of the comprehension system and its dependence on subjective judgement.

We observe, we regard from one or more points of view, we choose them among the millions that exist. [Tzara, 1918]

Appendix A

Example of Tree Creation

Important note: the example given in this appendix was generated using IDC's plan recognition schema set only. Note that the behaviour of the system changes if the whole schema set is used (see section 7.1.1).

In this appendix I give a more detailed example of IDC's tree creation. The representations shown are copied from the information produced by IDC during comprehension of a simple text. The text used is based on one of Ng's and is as follows:

Bill took a bus to a restaurant. [Ng and Mooney, 1992]

IDC's representation of this story consists of five observations:

```
event(e1, get_on, [agt:jack, loc_at:bs, pat:v]),
habit(bus, [exp:v]),
habit(bus_station, [exp:bs]),
event(e2, get_off, [agt:jack, loc_at:r, pat:v]),
habit(restaurant, [exp:r]).
```

The example below shows the transition from a state where IDC has read the first four observations but has not connected them together, to a state where it makes an inference about John's high-level plan.

The schema used by IDC connects the first four observations with a *go_by_bus* node. As IDC has a *get_off* and *bus* instances in focus, it uses this schema to create a new tree from the bottom-up, as shown in the *new representation*. Although the inference actually

increases the incoherence in the STS (as it introduces new r-elts and a new tree), it reduces the overall incoherence of the representation.

Some points to note:

- The inference is initiated because the incoherence in the initial representation's STS (= 4) is greater than IDC's tolerance (= 1.836). If no extension generated by creating a new tree had lowered incoherence, a transfer from the STS to the LTS would have been attempted (see section 6.4.1).
- All of the nodes in the schema which are not present in the representation are inferred as new instances. The schema is thus acting as a unit for retrieval and inference purposes and its activation is 'all-or-nothing'.
- If Skepticism is lowered to 0.1 but the other parameters left the same, IDC makes a high-level inference about a *go_by_vehicle* plan after reading only the first observation of this text. In other words, it jumps to the conclusion that John is going by vehicle, on the basis of him getting onto some thing *v* whose type is unknown. By contrast, with Skepticism = 0.5, this evidence alone is not considered strong enough to warrant inference of the high-level plan.

If Skepticism is raised to 0.9, IDC fails to make any high-level inferences: the *go_by_bus* event requires too many subsidiary events to be inferred, so IDC considers it safer to make none at all.

Parameter settings:*Skepticism* = 0.5*Tolerance* = 1.836*Range* = 0**Initial representation:***STS Instances:*

```
instance(@event(e2, get_off, [agt:jack, loc_at:r, pat:v]), n, n, 3.0),
instance(@habit(bus, [exp:v]), n, n, 1.0).
```

STS Forest:

no trees

LTS Instances:

```
instance(@habit(bus_station, [exp:bs]), n, n, 1.0),
instance(@event(e1, get_on, [agt:jack, loc_at:bs, pat:v]), n, n, 3.0).
```

LTS Forest:

no trees

Incoherence:

4 (STS); 4 (LTS)

Schema used to create new tree:

```
2: event(_, go_by_bus, [goer:A, bus:B, driver:C,
token:D, loc_from:E, loc_to:F]) / [] →
[event(_, go, [agt:A, loc_to:E]) / [],
event(_, get_on, [agt:A, loc_at:E, pat:B]) / [],
event(_, give, [agt:A, pat:C, obj:D]) / [],
event(_, sit, [agt:A, pat:G]) / [],
event(_, get_off, [agt:A, loc_at:F, pat:B]) / [F \== E],
habit(bus_station, [exp:H]) / [H ==> E],
habit(bus, [exp:I]) / [nonvar(I), I ==> B],
habit(vehicle_seat, [exp:G]) / [],
habit(bus_driver, [exp:C]) / [],
```

```

relation(part_of, [G, B]) / [],
habit(token, [exp:D]) / [] ].

```

New representation:*STS Instances:*

```

instance(@event(e2, get_off, [agt:jack, loc_at:r, pat:v]), y, n, 0.0),
instance(#event(_10719, go, [agt:jack, loc_to:bs]), y, n, 1.0),
instance(#event(_10692, give, [agt:jack, pat:_10683,
  obj:_10678]), y, n, 0.5),
instance(#event(_10660, sit, [agt:jack, pat:_10651]), y, n, 0.5),
instance(#habit(vehicle_seat, [exp:_10651]), y, n, 0.5),
instance(#habit(bus_driver, [exp:_10683]), y, n, 0.5),
instance(#relation(part_of, [_10651, v]), y, n, 0.5),
instance(#habit(token, [exp:_10678]), y, n, 0.5),
instance(#event(_10550, go_by_bus, [goer:jack, bus:v,
  driver:_10683, token:_10678, loc_from:bs, loc_to:r]), n, y, 0.5),
instance(@habit(bus, [exp:v]), y, n, 0.0).

```

STS Forest:

```

tree(2, #event(_10550, go_by_bus, [goer:jack,
  bus:v, driver:_10683, token:_10678, loc_from:bs, loc_to:r]),
  [#event(_10719, go, [agt:jack, loc_to:bs]),
    @event(e1, get_on, [agt:jack, loc_at:bs, pat:v]),
    #event(_10692, give, [agt:jack, pat:_10683, obj:_10678]),
    #event(_10660, sit, [agt:jack, pat:_10651]),
    @event(e2, get_off, [agt:jack, loc_at:r, pat:v]),
    @habit(bus_station, [exp:bs]),
    @habit(bus, [exp:v]),
    #habit(vehicle_seat, [exp:_10651]),
    #habit(bus_driver, [exp:_10683]),
    #relation(part_of, [_10651, v]),
    #habit(token, [exp:_10678])], 0.0).

```

LTS Instances:

```

instance(@habit(bus_station, [exp:bs]), y, n, 0.0),
instance(@event(e1, get_on, [agt:jack, loc_at:bs, pat:v]), y, n, 0.0).

```

LTS Forest:

no trees

Incoherence:

4.5 (STS); 0 (LTS)

Incoherence change:

$$(4 + 4) - (4.5 + 0) = -3.5$$

Appendix B

IDC Schema Code, Texts and Examples

B.1 Plan Recognition

B.1.1 Plan Recognition Schemas

```
event(_, go_by_vehicle, [goer:A, vehicle:V, loc_from:P,  
loc_to:D]) / [] --->
```

```
[  
  event(_, get_on, [agt:A, loc_at:P, pat:V]) / [],  
  event(_, sit, [agt:A, pat:S]) / [],  
  event(_, get_off, [agt:A, loc_at:D, pat:V]) / [],  
  habit(vehicle, [exp:V]) / [],  
  habit(vehicle_seat, [exp:S]) / [],  
  relation(part_of, [S, V]) / []  
].
```

```
event(_, go_by_bus, [goer:A, bus:V2, driver:R, token:T,  
loc_from:P2, loc_to:D]) / [] --->
```

```
[  
  event(_, go, [agt:A, loc_to:P2]) / [],  
  event(_, get_on, [agt:A, loc_at:P2, pat:V2]) / [],  
  event(_, give, [agt:A, pat:R, obj:T]) / [],
```



```

event(_, sit, [agt:A, pat:S]) / [],
event(_, get_off, [agt:A, loc_at:D, pat:V2]) / [D \== P2],
habit(bus_station, [exp:P1]) / [P1 ==> P2],
habit(bus, [exp:V1]) / [nonvar(V1), V1 ==> V2],
habit(vehicle_seat, [exp:S]) / [],
habit(bus_driver, [exp:R]) / [],
relation(part_of, [S, V2]) / [],
habit(token, [exp:T]) / []
].

```

```

event(_, go_by_taxi, [goer:A, taxi:V2, driver:R,
loc_from:P, loc_to:D]) / [] --->
[
event(_, go, [agt:A, loc_to:P]) / [],
event(_, get_on, [agt:A, loc_at:P, pat:V2]) / [],
event(_, sit, [agt:A, pat:S]) / [],
event(_, pay_step, [payer:A, payee:R, _]) / [],
event(_, get_off, [agt:A, loc_at:D, pat:V2]) / [D \== P],
habit(vehicle_seat, [exp:S]) / [],
habit(taxi, [exp:V1]) / [nonvar(V1), V1 ==> V2],
habit(taxi_driver, [exp:R]) / [],
relation(part_of, [S, V2]) / []
].

```

```

event(_, supermarket_shopping, [shopper:S, store:T2,
thing_bought:B]) / [] --->
[
event(_, shopping, [shopper:S, store:T2,
thing_bought:B]) / [],
habit(super_market, [exp:T1]) / [T1 ==> T2],
habit(food, [exp:B]) / []
].

```

```

event(_, liquor_store_shopping, [shopper:S, store:T2,
thing_bought:B2]) / [] --->
[
event(_, shopping, [shopper:S, store:T2,

```

```

    thing_bought:B2]) / [],
    habit(liquor_store, [exp:T1]) / [T1 ==> T2],
    habit(liquor, [exp:B1]) / [nonvar(B1), B1 ==> B2]
  ].

/*
% Liquor-store shopping alternative schema

event(_, liquor_store_shopping, [shopper:S, store:T2,
thing_bought:B]) / [] --->
  [
    event(_, shopping, [shopper:S, store:T2,
thing_bought:B]) / [],
    habit(liquor_store, [exp:T1]) / [T1 ==> T2],
    habit(liquor, [exp:B]) / []
  ].

*/

event(_, shopping, [shopper:S, store:T,
thing_bought:B]) / [] --->
  [
    event(_, go_step, [agt:S, loc_to:T]) / [],
    event(_, find, [agt:S, pat:B]) / [],
    event(_, buy_step, [buyer:S, bought:B]) / [],
    habit(shopping_place, [exp:T]) / []
  ].

event(_, rest_visit, [diner:A, restaurant:R2,
thing_ordered:0, utensils:I]) / [] --->
  [
    event(_, dining, [diner:A, place:R2,
thing_ordered:0, utensils:I]) / [],
    habit(restaurant, [exp:R1]) / [R1 ==> R2]
  ].

event(_, dining, [diner:A, place:R, thing_ordered:0,
utensils:I]) / [] --->

```

```

    [
      event(_, go_step, [agt:A, loc_to:R]) / [],
      event(_, order, [agt:A, pat:0]) / [],
      event(_, ingest_step, [agt:A, ingested:0, inst:I]) / [],
      event(_, pay_step, [payer:A, _, paid_for:0]) / [],
      habit(eating_place, [exp:R]) / []
    ].

event(_, robbing, [robber:A, weapon_used:W, place_robbed:P2,
thing_robbed:V, victim:M]) / [V \== W] --->
    [
      event(_, get, [agt:A, pat:W, _]) / [],
      event(_, go_step, [agt:A, loc_to:P2]) / [],
      event(_, point, [agt:A, pat:W, obj:M]) / [],
      event(_, get, [agt:A, pat:V, from:M]) / [],
      habit(valuable, [exp:V]) / [],
      habit(business, [exp:P1]) / [P1 ==> P2],
      habit(weapon, [exp:W]) / [],
      habit(in_charge, [agt:M, pat:P2]) / []
    ].

/*
% Mugging schema

event(_, mugging, [mugger:A, weapon_used:W, thing_robbed:V,
victim:M]) / [] --->
    [
      event(_, point, [agt:A, pat:W, obj:M]) / [],
      event(_, get, [agt:A, pat:V, from:M]) / [],
      habit(valuable, [exp:V]) / [],
      habit(weapon, [exp:W]) / []
    ].

*/

habit(owns, [agt:M, pat:P]) / [] --->
    [habit(in_charge, [agt:M, pat:P]) / []].

```



```
% ISA HIERARCHY
```

```
habit(liquor-store, [exp:P1]) / [P1 ==> P2] ---->  
    [habit(store, [exp:P2]) / []].
```

```
habit(supermarket, [exp:P1]) / [P1 ==> P2] ---->  
    [habit(store, [exp:P2]) / []].
```

```
habit(store, [exp:P1]) / [P1 ==> P2] ---->  
    [habit(shopping_place, [exp:P2]) / []].
```

```
habit(shopping_place, [exp:P1]) / [P1 ==> P2] ---->  
    [habit(business, [exp:P2]) / []].
```

```
habit(restaurant, [exp:P1]) / [P1 ==> P2] ---->  
    [habit(eating_place, [exp:P2]) / []].
```

```
habit(eating_place, [exp:P1]) / [P1 ==> P2] ---->  
    [habit(business, [exp:P2]) / []].
```

```
habit(bank, [exp:P1]) / [P1 ==> P2] ---->  
    [habit(business, [exp:P2]) / []].
```

```
habit(gun, [exp:W1]) / [W1 ==> W2] ---->  
    [habit(weapon, [exp:W2]) / []].
```

```
habit(sword, [exp:W1]) / [W1 ==> W2] ---->  
    [habit(weapon, [exp:W2]) / []].
```

```
habit(milkshake, [exp:A]) / [] ---->  
    [habit(beverage, [exp:A]) / []].
```

```
habit(bread, [exp:E1]) / [E1 ==> E2] ---->  
    [habit(food, [exp:E2]) / []].
```

```
habit(bourbon, [exp:E1]) / [E1 ==> E2] ---->  
    [habit(liquor, [exp:E2]) / []].
```

```
habit(knife, [exp:W1]) / [W1 ==> W2] ---->
    [habit(weapon, [exp:W2]) / []].

habit(razor, [exp:W1]) / [W1 ==> W2] ---->
    [habit(weapon, [exp:W2]) / []].

habit(knife, [exp:W1]) / [W1 ==> W2] ---->
    [habit(sharp_object, [exp:W2]) / []].

habit(razor, [exp:W1]) / [W1 ==> W2] ---->
    [habit(sharp_object, [exp:W2]) / []].

habit(sword, [exp:W1]) / [W1 ==> W2] ---->
    [habit(sharp_object, [exp:W2]) / []].

habit(money, [exp:V1]) / [V1 ==> V2] ---->
    [habit(valuable, [exp:V2]) / []].

habit(antique, [exp:V1]) / [V1 ==> V2] ---->
    [habit(valuable, [exp:V2]) / []].

habit(banana, [exp:F1]) / [F1 ==> F2] ---->
    [habit(food, [exp:F2]) / []].

habit(potato, [exp:F1]) / [F1 ==> F2] ---->
    [habit(food, [exp:F2]) / []].

habit(beef, [exp:F1]) / [F1 ==> F2] ---->
    [habit(food, [exp:F2]) / []].

habit(tomato, [exp:F1]) / [F1 ==> F2] ---->
    [habit(food, [exp:F2]) / []].

habit(chips, [exp:F1]) / [F1 ==> F2] ---->
    [habit(food, [exp:F2]) / []].
```

B.1.2 Example Plan Recognition Trees

This section contains examples of trees constructed by IDC during comprehension of the liquor-store text of section 7.1.1 (page 159). This text is repeated here for convenience:

```
[event(e1, go, [agt:bob, loc_to:ls]), habit(liquor_store, [exp:ls]), event(e2, point,
[agt:bob, pat:w, obj:o]), habit(gun, [exp:w]), habit(owns, [agt:o, pat:ls])]
```

A rough English version of each tree is given, then the Prolog structure generated by IDC. The format of the trees follows the description given in section 5.3.5 (page 118); no incoherence values are given for the trees, as this varies according to parameter settings. I have replaced Prolog's obscure internal variables with uninstantiated variables (capital letters); it is easy to track the embedding of one plan inside another by matching instances in one tree with those of another tree.

The trees *actually* inferred by IDC under various parameter settings are shown in section 7.1.1.

It is worth noting that IDC may infer certain instances which are not appropriately merged with other instance. For example, when Skepticism is low (= 0.1), IDC infers a *robbing* event without tying it to the *go* event. This results in the construction of a *go_step* instance which is distinct from the *go_step* previously inferred from the *go* event. An example of such a situation is shown in bullet point 5.

1. *Bob went liquor-store shopping.*

```
tree(5, #event(M, liquor_store_shopping, [shopper:bob, store:ls,
thing_bought:B]),
[#event(E, shopping, [shopper:bob, store:ls, thing_bought:B]),
@habit(liquor_store, [exp:ls]), #habit(liquor, [exp:B])])
```

2. *Bob went shopping in a shopping place (the liquor-store).*

```
tree(6, #event(E, shopping, [shopper:bob, store:ls, thing_bought:B]),
[#event(F, go_step, [agt:bob, loc_to:ls]),
#event(G, find, [agt:bob, pat:B]),
#event(A, buy_step, [buyer:bob, bought:B]),
#habit(shopping_place, [exp:ls])])
```

3. *Bob's going isa go-step.*

```
tree(19, @event(e1, go, [agt:bob, loc_to:ls]),
      [#event(F, go_step, [agt:bob, loc_to:ls])])
```

4. *Bob robbed the liquor-store using the gun; the victim of the crime was the owner of the store.*

```
tree(9, #event(H, robbing, [robber:bob, weapon_used:w, place_robbed:ls,
thing_robbed:I, victim:o]),
      [#event(J, get, [agt:bob, pat:w, from:K]),
       #event(F, go_step, [agt:bob, loc_to:ls]),
       @event(e2, point, [agt:bob, pat:w, obj:o]),
       #event(L, get, [agt:bob, pat:I, from:o]),
       #habit(valuable, [exp:I]), #habit(business, [exp:ls]),
       #habit(weapon, [exp:w]), #habit(in_charge, [agt:o, pat:ls])])
```

5. *Bob robbed the liquor-store using the gun; the victim of the crime was the owner of the store.*

```
tree(9, #event(M, robbing, [robber:bob, weapon_used:w, place_robbed:ls,
thing_robbed:I, victim:o]),
      [#event(J, get, [agt:bob, pat:w, from:K]),
       #event(N, go_step, [agt:bob, loc_to:ls]),
       @event(e2, point, [agt:bob, pat:w, obj:o]),
       #event(L, get, [agt:bob, pat:I, from:o]),
       #habit(valuable, [exp:I]), #habit(business, [exp:ls]),
       #habit(weapon, [exp:w]), #habit(in_charge, [agt:o, pat:ls])])
```

(Note that the *go_step* here is distinct from the *go_step* of bullet point 4.)

6. *The shopping-place (the liquor-store) is a business.*

```
tree(23, #habit(shopping_place, [exp:ls]),
      [#habit(business, [exp:ls])])
```

7. *The gun is a weapon.*

```
tree(27, @habit(gun, [exp:w]),
      [#habit(weapon, [exp:w])])
```

8. *The owner of the liquor-store is in charge of it.*


```
tree(10, @habit(owns, [agt:o, pat:ls]),  
      [#habit(in_charge, [agt:o, pat:ls])])
```

N.B. The first two trees, while superficially similar in their English translations, actually depict two different hypotheses: the second denotes a general shopping plan at some 'shopping place', while the first is a specialised shopping plan, specifically related to liquor shopping.

B.2 Inference Protocols

B.2.1 Ivan Story Text

This section contains the original text and the IDC version of ‘The Ivan Story’ (based on [Trabasso and Magliano, 1996]). IDC’s comprehension of this text is described in section 7.2.

Original text of ‘The Ivan Story’

- S1. Ivan was a great warrior.
- S2. Ivan was the best archer in his village.
- S3. Ivan heard that a giant was terrifying the people in his village.
- S4. The giant came to the village at night and hurt people.
- S5. Ivan was determined to kill the giant.
- S6. Ivan waited until dark.
- S7. The giant came and Ivan shot an arrow at him.
- S8. Ivan hit the giant and the giant fell down.
- S9. The people were overjoyed.

(adapted from Trabasso and Magliano [Trabasso and Magliano, 1996])

IDC version of ‘The Ivan Story’

```

habit(great_warrior, [exp:ivan]),
habit(best_archer, [exp:ivan, loc_in:village]),
habit(live, [exp:ivan, loc_in:village]),
event(e1, terrorise, [agt:giant, pat:people]),
habit(live, [exp:people, loc_in:village]),
event(e2, come, [agt:giant, loc_to:village, when:night]),
event(e3, hurt, [agt:giant, pat:people, inst:-]),
relation(precedes, [e2, e3]),
goal(g1, kill, [agt:ivan, pat:giant]),
relation(overlaps, [e3, g1]),
event(e4, wait_for, [agt:ivan, pat:giant, loc_at:village, when:night]),
relation(overlaps, [g1, e4]),
event(e5, arrive, [agt:giant, loc_at:village, when:night]),
relation(overlaps, [e4, e5]),
event(e6, shot, [agt:ivan, pat:giant, inst:a]),

```

```

habit(arrows, [exp:a]),
relation(precedes, [e5,e6]),
event(e7, hit, [agt:ivan, pat:giant, inst:a]),
relation(precedes, [e6,e7]),
event(e8, fall, [agt:giant, loc_to:ground]),
relation(precedes, [e7,e8]),
event(e9, overjoy, [agt:people]),
relation(precedes, [e8,e9]).

```

B.2.2 Ivan Story Schemas

```

habit(good_fighter, [exp:A]) / [] --->
    [habit(great_warrior, [exp:A]) / []].

```

```

habit(archer, [exp:A]) / [X2 \== Y2, X2 \== Z2, Y2 \== Z2] --->
    [
    habit(armour, [exp:X1]) / [X1 ==> X2],
    habit(bow, [exp:Y1]) / [Y1 ==> Y2],
    habit(arrows, [exp:Z1]) / [Z1 ==> Z2],
    habit(use, [agt:A, inst:X2]) / [],
    habit(use, [agt:A, inst:Y2]) / [],
    habit(use, [agt:A, inst:Z2]) / []
    ].

```

```

habit(best_archer, [exp:A1, _]) / [A1 ==> A2] --->
    [habit(archer, [exp:A2]) / []].

```

```

event(E4a, protect, [agt:Champion1, pat:Villagers1, from:Threat]) /
[E4a ==> E4b, Champion1 ==> Champion2, Villagers1 ==> Villagers2] --->
    [
    event(E1a, come, [agt:Threat, loc_to:Place1, when:_]) /
        [E1a ==> E1b, Place1 ==> Place2],
    event(E2a, hurt, [agt:Threat, pat:Villagers1, inst:_]) /
        [E2a ==> E2b],
    goal(G1a, kill, [agt:Champion1, pat:Threat]) / [G1a ==> G1b],

```

```

event(E3a, fight, [agt:Champion1, pat:Threat]) / [E3a ==> E3b],
habit(care_about, [exp:Champion2, exp:Villagers1]) / [],
habit(live, [exp:Villagers2, loc_in:Place2]) / [],
habit(good_fighter, [exp:Champion2]) / [],
relation(phycs, [E1b, E2b]) / [],
relation(phycs, [E2b, G1b]) / [],
relation(motivates, [G1b, E3b]) / [],
relation(includes, [E4b, [E1b, E2b, G1b, E3b]]) / []
].

```

```

relation(phycs, [E1b, E2b]) / [] --->
[
event(E1a, come, [agt:Threat, loc_to:Place1, when:_]) /
[E1a ==> E1b, Place1 ==> Place2],
event(E2a, hurt, [agt:Threat, pat:Villagers, inst:_]) /
[E2a ==> E2b],
habit(live, [exp:Villagers, loc_in:Place2]) / [],
relation(precedes, [E1b, E2b]) / []
].

```

```

relation(psycs, [E1b, G1b]) / [] --->
[
event(E1a, hurt, [agt:Threat, pat:People, inst:_]) /
[E1a ==> E1b],
goal(G1a, kill, [agt:Champion, pat:Threat]) / [G1a ==> G1b],
habit(care_about, [exp:Champion, exp:People]) / [],
relation(overlaps, [E1b, G1b]) / []
].

```

```

relation(motivates, [G1b, E1b]) / [] --->
[
goal(G1a, kill, [agt:Champion, pat:Threat]) / [G1a ==> G1b],
event(E1a, fight, [agt:Champion, pat:Threat]) / [E1a ==> E1b],
relation(overlaps, [G1b, E1b]) / []
].

```

```

habit(care_about, [exp:A2, exp:B2]) / [A2 \== B2] --->

```

```

[
  habit(live, [exp:A1, loc_in:L]) / [A1 ==> A2],
  habit(live, [exp:B1, loc_in:L]) / [B1 ==> B2]
].

```

```

relation(psycs, [E1b, E2b]) / [] --->
[
  event(E1a, terrorise, [agt:_, pat:Friend2a]) /
    [E1a ==> E1b, Friend2a ==> Friend2b],
  event(E2a, anger, [exp:Friend1a]) /
    [E2a ==> E2b, Friend1a ==> Friend1b],
  habit(care_about, [exp:Friend1b, exp:Friend2b]) / [],
  relation(precedes, [E1b, E2b]) / []
].

```

```

relation(psycs, [E1b, G1b]) / [] --->
[
  event(E1a, terrorise, [agt:Threat, pat:People]) /
    [E1a ==> E1b],
  goal(G1a, kill, [agt:Champion, pat:Threat]) / [G1a ==> G1b],
  habit(care_about, [exp:Champion, exp:People]) / [],
  relation(overlaps, [E1b, G1b]) / []
].

```

```

habit(villagers, [exp:P1]) / [] --->
[
  habit(people, [exp:P1]) / [P1 ==> P2],
  habit(village, [exp:V1]) / [V1 ==> V2],
  habit(live, [exp:P2, loc_in:V2]) / []
].

```

```

event(E1, stab, [agt:A, pat:B, inst:K1]) / [K1 ==> K2] --->
[
  event(E2, hurt, [agt:A, pat:B, inst:K2]) / [K1 ==> K2],
  habit(knife, [exp:K1]) / [],
  relation(simultaneous, [E1, E2]) / []
].

```

```

relation(phycs, [E1b, E2b]) / [] --->
    [
    event(E1a, hit, [agt:_, pat:B]) / [E1a ==> E1b],
    event(E2a, pain, [exp:B]) / [E2a ==> E2b],
    relation(precedes, [E1b, E2b]) / []
    ].

relation(phycs, [E1b, E2b]) / [] --->
    [
    event(E1a, strike, [agt:A, pat:B1]) / [E1a ==> E1b],
    event(E2a, light, [agt:A, exp:B1]) / [E2a ==> E2b],
    habit(match, [exp:B2]) / [B1 ==> B2],
    relation(precedes, [E1b, E2b]) / []
    ].

habit(care_about, [exp:A2, exp:B2]) / [] --->
    [habit(at_uni_together, [exp:A1, exp:B1]) /
    [A1 ==> A2, B1 ==> B2]].

habit(care_about, [exp:A2, exp:B2]) / [] --->
    [habit(brothers, [exp:A1, exp:B1]) / [A1 ==> A2, B1 ==> B2]].

relation(phycs, [E1b, E2b]) / [] --->
    [
    event(E1a, in_trouble, [exp:B1]) / [E1a ==> E1b],
    event(E2a, help, [agt:A1, pat:B1]) /
    [E2a ==> E2b, A1 ==> A2, B1 ==> B2],
    habit(care_about, [exp:A2, exp:B2]) / [],
    relation(overlaps, [E1b, E2b]) / []
    ].

event(E1a, terrorise, [agt:Threat, pat:People]) / [E1a ==> E1b] --->
    [
    event(E2a, attack, [agt:Threat, pat:People, inst:_, when:_]) /
    [E2a ==> E2b],
    relation(simultaneous, [E1b, E2b]) / []
    ]

```

].

```
relation(phycs, [E1b, E2b]) / [] --->
[
  event(E1a, attack, [agt:_, pat:P, inst:_, when:_]) /
    [E1a ==> E1b],
  event(E2a, in_trouble, [exp:P]) / [E2a ==> E2b],
  relation(precedes, [E1b, E2b]) / []
].
```

```
relation(enables, [E1b, E2b]) / [] --->
[
  event(E1a, come, [agt:A, loc_to:P, when:_]) / [E1a ==> E1b],
  event(E2a, at, [agt:A, loc:P]) / [E2a ==> E2b],
  relation(precedes, [E1b, E2b]) / []
].
```

```
relation(psycs, [E1b, E2b]) / [] --->
[
  event(E1a, terrorise, [agt:A, pat:P]) / [E1a ==> E1b],
  event(E2a, fear, [agt:P, obj:A]) / [E2a ==> E2b],
  relation(precedes, [E1b, E2b]) / []
].
```

```
habit(know, [agt:A2, pat:B2]) / [] --->
[
  habit(live, [agt:A1, loc_in:L]) / [A1 ==> A2],
  habit(live, [agt:B1, loc_in:L]) / [B1 ==> B2]
].
```

```
event(E1a, successful_goal, [agt:A]) / [E1a ==> E1b] --->
[
  goal(G1a, kill, [agt:A, pat:P]) / [G1a ==> G1b],
  event(E2a, kill, [agt:A, pat:P]) / [E2a ==> E2b],
  relation(overlaps, [G1b, E2b]) / [],
  relation(overlaps, [E2b, E1b]) / []
].
```

```

relation(motivates, [G1b, E2b]) / [] --->
  [
    goal(G1a, hide_from, [agt:A, loc_at:L, pat:_]) / [G1a ==> G1b],
    event(E2a, wait, [agt:A, loc_at:L, when:night]) / [E2a ==> E2b],
    relation(overlaps, [G1b, E2b]) / []
  ].

```

```

relation(motivates, [G1b, G2b]) / [] --->
  [
    goal(G1a, kill, [agt:A, pat:P]) / [G1a ==> G1b],
    goal(G2a, hide_from, [agt:A, pat:P]) / [G2a ==> G2b],
    relation(overlaps, [G1b, G2b]) / []
  ].

```

```

relation(enables, [E1b, E2b]) / [] --->
  [
    event(E1a, come, [agt:A, loc_to:P, when:Time]) / [E1a ==> E1b],
    event(E2a, wait_for, [agt:B, pat:A, loc_at:P, when:Time]) /
      [E2a ==> E2b, B \== A],
    relation(precedes, [E1b, E2b]) / []
  ].

```

```

event(E1a, fight, [agt:Champion, pat:Threat]) / [E1a ==> E1b] --->
  [
    event(E2a, shot, [agt:Champion, pat:Threat, inst:I1]) /
      [E2a ==> E2b, I1 ==> I2],
    habit(weapon, [exp:I2]) / [],
    relation(simultaneous, [E1b, E2b]) / []
  ].

```

```

relation(motivates, [G1b, E1b]) / [] --->
  [
    goal(G1a, kill, [agt:A, pat:P]) / [G1a ==> G1b],
    event(E1a, shot, [agt:A, pat:P, inst:I1]) /
      [E1a ==> E1b, I1 ==> I2],
    habit(weapon, [exp:I2]) / [],

```



```

relation(overlaps, [G1b, E1b]) / []
].

```

```

event(E1a, ambush, [agt:A, pat:B]) / [E1a ==> E1b] --->
[
event(E2a, wait_for, [agt:A, pat:B, loc_at:P, when:Time]) /
[E2a ==> E2b],
event(E3a, arrive, [agt:B, loc_at:P, when:Time]) /
[E3a ==> E3b],
event(E4a, fight, [agt:A, pat:B, inst:I1]) /
[E4a ==> E4b, I1 ==> I2],
habit(weapon, [exp:I2]) / [],
relation(precedes, [E2b, E3b]) / [],
relation(precedes, [E3b, E4b]) / [],
relation(includes, [E1b, [E2b, E3b, E4b]]) / []
].

```

```

relation(phycs, [E1b, E2b]) / [] --->
[
event(E1a, shot, [agt:A, pat:P, inst:I]) / [E1a ==> E1b],
event(E2a, hit, [agt:A, pat:P, inst:I]) / [E2a ==> E2b],
relation(precedes, [E1b, E2b]) / []
].

```

```

event(E1a, kill, [agt:A, pat:P]) / [E1a ==> E1b] --->
[
event(E2a, shot, [agt:A, pat:P, inst:I]) / [E2a ==> E2b],
event(E3a, hit, [agt:A, pat:P, inst:I]) / [E3a ==> E3b],
event(E4a, fall, [agt:P, loc_to:_]) / [E4a ==> E4b],
relation(precedes, [E2b, E3b]) / [],
relation(precedes, [E3b, E4b]) / [],
relation(overlaps, [E4b, E1b]) / []
].

```

```

relation(enables, [E1b, E2b]) / [] --->
[
event(E1a, overjoy, [agt:A]) / [E1a ==> E1b],

```

```

event(E2a, celebrate, [agt:A]) / [E2a ==> E2b],
relation(overlaps, [E1b, E2b]) / []
].

```

```

relation(phycs, [E2b, E3b]) / [] --->
[
event(E1a, fear, [agt:A, obj:P]) / [E1a ==> E1b],
event(E2a, kill, [agt:_, pat:P]) / [E2a ==> E2b],
event(E3a, no_fear, [agt:A, obj:P]) / [E3a ==> E3b],
relation(precedes, [E1b, E2b]) / [],
relation(precedes, [E2b, E3b]) / []
].

```

```

relation(psycs, [E2b, E3b]) / [] --->
[
event(E1a, fear, [agt:A, obj:P]) / [E1a ==> E1b],
event(E2a, no_fear, [agt:_, pat:P]) / [E2a ==> E2b],
event(E3a, overjoy, [agt:A]) / [E3a ==> E3b],
relation(precedes, [E1b, E2b]) / [],
relation(precedes, [E2b, E3b]) / []
].

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ISA HIERARCHY

```

```

habit(gun, [exp:W1]) / [W1 ==> W2] --->
[habit(weapon, [exp:W2]) / []].

```

```

habit(sword, [exp:W1]) / [W1 ==> W2] --->
[habit(weapon, [exp:W2]) / []].

```

```

habit(arrows, [exp:A1]) / [A1 ==> A2] --->
[habit(weapon, [exp:A2]) / []].

```

```

habit(knife, [exp:W1]) / [W1 ==> W2] --->
[habit(weapon, [exp:W2]) / []].

```

```
habit(razor, [exp:W1]) / [W1 ==> W2] --->  
  [habit(weapon, [exp:W2]) / []].
```

B.3 Role Shift Texts

B.3.1 Janitor Story Text

This section contains the IDC versions of three Janitor Story texts (based on [Sanford and Garrod, 1981]):

1. The full Janitor Story.
2. The shortened role-shift version of the Janitor Story.
3. The shortened non-role-shift version of the Janitor Story.

IDC's comprehension of these texts is described in section 7.3.

1. Full Janitor Story: English text

John was on his way to school.

He was worried about the maths lesson.

Last week he lost control of the class.

It was unfair of the maths teacher to leave him in charge.

After all, teaching wasn't part of a janitor's duties.

Full Janitor Story: IDC version

event(e1, go, [agt:john, loc_to:s]),

habit(school, [exp:s]),

event(e2, worry_about, [agt:john, pat:m1]),

habit(math_lesson, [exp:m1]),

relation(precedes, [e1, e2]),

event(e3, lost_control, [agt:john, pat:m2]),

habit(math_lesson, [exp:m2]),

relation(precedes, [e3, e1]),

relation(precedes, [e3, e2]),

event(e4, put_in_charge, [agt:t, pat1:john, pat2:m2]),

habit(teacher, [exp:t]),

relation(precedes, [e4, e3]),

habit(janitor, [exp:john]).

2. Non-role-shift Janitor Story: English text

John was worried about teaching maths.
 He was on his way to school.
 Last week he lost control of the class.

Non-role-shift Janitor Story: IDC version

```
event(e2, worry_about, [agt:john, pat:m1]),
habit(teacher, [exp:john]),
habit(math_lesson, [exp:m1]),
event(e1, go, [agt:john, loc_to:s]),
habit(school, [exp:s]),
relation(precedes, [e2, e1]),
event(e3, lost_control, [agt:john, pat:m2]),
habit(math_lesson, [exp:m2]),
relation(precedes, [e3, e1]),
relation(precedes, [e3, e2]).
```

3. Role-shift Janitor Story: English text

John was on his way to school.
 He was worried about the maths lesson.
 Last week he lost control of the class.

Role-shift Janitor Story: IDC version

```
event(e1, go, [agt:john, loc_to:s]),
habit(school, [exp:s]),
event(e2, worry_about, [agt:john, pat:m1]),
habit(math_lesson, [exp:m1]),
relation(precedes, [e1, e2]),
event(e3, lost_control, [agt:john, pat:m2]),
habit(math_lesson, [exp:m2]),
relation(precedes, [e3, e1]),
relation(precedes, [e3, e2]).
```

B.3.2 Janitor Story Schemas

```
relation(motivates, [G1b, E2b]) / [] --->
[
```

```

goal(G1a, learn, [exp:A, loc_at:S2]) / [G1a ==> G1b],
event(E2a, go, [agt:A, loc_to:S2]) / [E2a ==> E2b],
habit(school, [exp:S1]) / [S1 ==> S2],
habit(schoolchild, [exp:A]) / []
].

```

```

relation(motivates, [G1b, E2b]) / [] --->
[
goal(G1a, teach, [exp:A, loc_at:S2]) / [G1a ==> G1b],
event(E2a, go, [agt:A, loc_to:S2]) / [E2a ==> E2b],
habit(school, [exp:S1]) / [S1 ==> S2],
habit(teacher, [exp:A]) / []
].

```

```

relation(motivates, [G1b, E2b]) / [] --->
[
goal(G1a, earn_living, [exp:A, loc_at:S2]) / [G1a ==> G1b],
event(E2a, go, [agt:A, loc_to:S2]) / [E2a ==> E2b],
habit(school, [exp:S1]) / [S1 ==> S2],
habit(janitor, [exp:A]) / []
].

```

```

habit(schoolchild, [exp:A]) / [] --->
[
habit(attend, [agt:A, exp:S]) / [],
habit(school, [exp:S]) / []
].

```

```

habit(teacher, [exp:A]) / [] --->
[
habit(attend, [agt:A, exp:S]) / [],
habit(school, [exp:S]) / []
].

```

```

habit(janitor, [exp:A]) / [] --->
[
habit(attend, [agt:A, exp:S]) / [],

```

```
habit(school, [exp:S]) / []
].
```

```
event(E1a, afraid_of_professional_failure, [agt:A2]) / [E1a ==> E1b] --->
[
  event(E2a, worry_about, [agt:A1, pat:M1]) /
    [E2a ==> E2b, A1 ==> A2],
  event(E3a, lost_control, [agt:A1, pat:M2]) /
    [E3a ==> E3b, M2 \== M1],
  habit(lesson, [exp:M1]) / [],
  habit(lesson, [exp:M2]) / [],
  habit(teacher, [exp:A2]) / [],
  relation(precedes, [E3b, E2b]) / [],
  relation(simultaneous, [E1b, E2b]) / []
].
```

```
event(E1a, afraid_of_test, [agt:A2, pat:T2]) / [E1a ==> E1b] --->
[
  event(E2a, worry_about, [agt:A1, pat:M1]) /
    [E2a ==> E2b, A1 ==> A2, M1 ==> M2],
  event(E3a, fail, [agt:A1, thm:T1]) / [E3a ==> E3b, T1 ==> T2],
  habit(lesson, [exp:M2]) / [],
  habit(test, [exp:T2]) / [],
  habit(schoolchild, [exp:A2]) / [],
  relation(part_of, [T2, M2]) / [],
  relation(simultaneous, [E1b, E2b]) / [],
  relation(precedes, [E2b, E3b]) / []
].
```

```
event(E1a, afraid_of_losing_control_again, [agt:B2]) / [E1a ==> E1b] --->
[
  event(E2a, put_in_charge, [agt:A1, pat1:B1, pat2:L1]) /
    [E2a ==> E2b, A1 ==> A2, B1 ==> B2, L1 ==> L2],
  event(E3a, lost_control, [agt:B1, pat:L1]) / [E3a ==> E3b],
  event(E4a, worry_about, [agt:B1, pat:M1]) /
    [E4a ==> E4b, M1 ==> M2],
  habit(lesson, [exp:L2]) / [L2 \== M2],
].
```

```

habit(lesson, [exp:M2]) / [L2 \== M2],
habit(teacher, [exp:A2]) / [],
habit(replacement_teacher, [exp:B2]) / [B2 \== A2],
relation(precedes, [E2b, E3b]) / [],
relation(precedes, [E3b, E4b]) / [],
relation(simultaneous, [E1b, E4b]) / []
].

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ISA HIERARCHY

```

```

habit(english_lesson, [exp:M1]) / [M1 ==> M2] ---->
[habit(lesson, [exp:M2]) / []].

```

```

habit(math_lesson, [exp:M1]) / [M1 ==> M2] ---->
[habit(lesson, [exp:M2]) / []].

```

```

habit(geography_lesson, [exp:M1]) / [M1 ==> M2] ---->
[habit(lesson, [exp:M2]) / []].

```

```

habit(history_lesson, [exp:M1]) / [M1 ==> M2] ---->
[habit(lesson, [exp:M2]) / []].

```

```

habit(parent, [exp:A2]) / [] ---->
[habit(replacement_teacher, [exp:A1]) / [A1 ==> A2]].

```

```

habit(schoolchild, [exp:A2]) / [] ---->
[habit(replacement_teacher, [exp:A1]) / [A1 ==> A2]].

```

```

habit(janitor, [exp:A2]) / [] ---->
[habit(replacement_teacher, [exp:A1]) / [A1 ==> A2]].

```

```

habit(teacher, [exp:A2]) / [] ---->
[habit(replacement_teacher, [exp:A1]) / [A1 ==> A2]].

```

```

habit(teacher, [exp:A1]) / [A1 ==> A2] ---->
[habit(adult, [exp:A2]) / []].

```



```
habit(janitor, [exp:A1]) / [A1 ==> A2] ---->  
[habit(adult, [exp:A2]) / []].
```

```
habit(parent, [exp:A1]) / [A1 ==> A2] ---->  
[habit(adult, [exp:A2]) / []].
```

```
habit(schoolchild, [exp:A1]) / [A1 ==> A2] ---->  
[habit(child, [exp:A2]) / []].
```

Appendix C

IDC Program Code

C.1 Running IDC

IDC was implemented in SICStus Prolog, version 3.8; the code is available to interested parties. Installation should be fairly simple, mainly involving copying the IDC files into a dedicated directory called ‘IDC_dir’. This directory should contain a subdirectory ‘Output_dir’ for storing files produced by the *file_comp* procedure (see below).

On UNIX, this directory should be attached to the user’s root directory, to minimise problems caused by incorrect paths (e.g. when *IDC_setup* writes informativities and ubiquities to the knowledge base). If using IDC on a PC, changes will have to be made across the various program files: each occurrence of `~/IDC_dir` should be replaced by `C:/IDC_dir/` (assuming IDC is installed on the C: drive. However, there should be no compatibility problems caused by the operating system, and the code should probably also run in older versions of SICStus Prolog. Compatibility with other flavours of Prolog has not been tested.

The files necessary for running IDC are:

- *IDC.pl*: core program.
- *IDC_setup.pl*: knowledge base derivation.
- *schemas.pl*: file containing hand-coded schema definitions.
- *texts.pl*: file of texts. Each text is encoded as a two place predicate of the form:

text(Identifier, ListOfTextStatements)

For example, a simple text representing ‘John goes to school’ might be:

```
text(1, [event(e1, go, [agt:john, loc_to:s]), habit(school, [exp:s])]).
```

The identifier is used in the top-level call to IDC (see below).

- *shared.pl*: utility procedures, some of which are shared between *IDC.pl* and *IDC_setup.pl*.
- *tests.pl*: this is an optional file which contains various test procedures: for example, there is a facility which allows all extensions of a single representation to be generated, regardless of their incoherence.

C.1.1 Preparing IDC’s Knowledge Base

The first step in running IDC is to set up the knowledge base using the *IDC_setup.pl* file. This program consults *schemas.pl*, creating indexes and assigning informativities and ubiquities to the nodes in the schema lattice. The results are written to the file *kb.pl*, which is then used by the core program (see next section).

The program runs automatically once it is loaded into the interpreter. The only adjustments the user may want to make may be changing the schema set accessed by IDC.

C.1.2 Running the Comprehension Simulation

Once the knowledge base has been prepared, the main *IDC.pl* program can be loaded into the Prolog interpreter. It is probably best to run the main program in a separate session from the setup program.

IDC has three ‘modes’ of operation:

1. *comp(TextIdentifier, Skepticism, Range, ToleranceFactor)*

This is the normal method I’ve used. *TextIdentifier* represents the first argument of a `text` clause in *tests.pl* (see above).

2. *filecomp(FilePrefix, TextIdentifier, Skepticism, Range, ToleranceFactor)*

This call works exactly as a call to *comp* (see above), but writes the output to the file specified by *FilePrefix*. *FilePrefix* is an atom consisting of alphanumeric characters and underscores (usually). The actual output filename is:

`'~/IDC_dir/Output_dir/' + FilePrefix + '.idc'`

So, if *FilePrefix* == *text2010101*, the file holding the output is called:

`~/IDC_dir/Output_dir/text2010101.idc`

3. *text_comp(Text, Skepticism, Range, ToleranceFactor)*

This call allows a text to be input directly, bypassing the need for a *texts.pl* file. However, this is impractical for anything but very small texts.

In all cases, the other arguments are instantiated as follows: *Skepticism* is a value greater than 0 but less than 1; *Range* is either a number ≥ 0 (for multiple representations), or -1 to maintain a single representation; *Tolerance* is any number ≥ 0 . (Note that the actual *Tolerance* of the comprehender is equal to the average informativity of nodes in the knowledge base multiplied by *ToleranceFactor*.)

IDC will always attempt to comprehend a text, regardless of the schemas in its knowledge base. It is worth bearing this in mind if it seems to be do nothing at all; however, it is worth pointing out that at high *Skepticism* settings, IDC can have a tendency to simply move r-elts around without making any inferences.

If the *filecomp* mode is used, the output file is a plain text file containing the entire session's content. This can be cumbersome to analyse in a text editor, so I have implemented a viewer for these files (written in Python). This can also be supplied with the rest of the program code.

The full code for the main IDC procedures constitutes the rest of this appendix.

C.2 *IDC_setup.pl*

```

/*****
IDC_SETUP

FILENAME:
['~/IDC_dir/idc_setup.pl'].

FILES USED:
schemas.pl
lists.pl

PURPOSE:
Creates the files used by idc.pl, namely:
    1. nodes.pl
       File containing node definitions, listing informativity
       and ubiquity.
    2. indices.pl
       File containing schema indexes.

METHOD:
Automatically creates files via a call of create_files/0.
*****/

% set up a gensym for assigning ID numbers to schemas
:- dynamic(id/1).
id(1).

% declare operator for schema construction
:- op(600, xfx, --->).

% operator for schema constraints
:- op(700, xfx, ==>).

% declare operator for inferred nodes
:- op(100, fyx, #).

% declare operator for observed nodes
:- op(100, fyx, @).

% QUICK RELOAD
rl :- ['~/IDC_dir/idc_setup.pl'].

```

```
% SHARED PROCEDURES
% (variable binding, constraint checking, output, lists)
:- ['~/IDC_dir/shared.pl'].

% FILE CONTAINING ALL SCHEMAS
:- ['~/IDC_dir/schemas.pl'].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% TOP-LEVEL SETUP PREDICATES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create_kb/0
% Create the two files to be used by the main program, idc.pl
% Both of these programs rely on the data in schemas.pl
% NB it is necessary to load the kb.pl file once generated, as
% it is used by define/3

create_kb :-
    process_raw_schemas(Indexes, AnnotatedSchemas),
    open('~/IDC_dir/kb.pl', write, File1),
    set_output(File1),
    write_clauses_to_stream(Indexes),
    nl, nl,
    close(File1),
    consult('~/IDC_dir/kb.pl'),
    define(NodesList, TotalInformativity, AverageInformativity),
    open('~/IDC_dir/kb.pl', append, File2),
    set_output(File2),
    write_clauses_to_stream(AnnotatedSchemas),
    nl, nl,
    write_clauses_to_stream([total_inf(TotalInformativity),
                             avg_inf(AverageInformativity) | NodesList]),
    close(File2),
    nl, nl,
    write_nodes(NodesList),
```



```

% Looks up the value in the id/1 clause, retracts that value,
% and updates it. Creates an ID for a schema by attaching to ID
% value to the string 's'.

make_id(Val) :-
    id(Val),
    retract(id(Val1)),
    Val2 is Val1 + 1,
    assert(id(Val2)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% process_all(Schemas, Indexes, AnnotatedSchemas)
% arg1 = list of lists of all schemas in knowledge base; each
% sublist represents a schema and has form
% schema(ID, LH, [RH1,..., RHn], Annotated).
% arg2 = list of indexes of those schemas.
% arg3 = list of schemas annotated with ID numbers.
% Creates index/3 clauses for all schemas in knowledge base,
% and the list of all schemas annotated with their IDs

process_all([], [], []).

process_all([schema(ID, LH/Cons, RHs) | RestSchemas], Indexes,
[ID:LH/Cons ---> RHs | RestAnnotated]) :-
    index_one(RHs, ID, LH, IndexSchema),
    process_all(RestSchemas, RestIndexes, RestAnnotated),
    append(IndexSchema, RestIndexes, Indexes).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% index_one(RHs, SchemaID, LH, SchemaIndexes)
% where LH is the left-hand node of the schema,
% RHs is the set of right-hand nodes of that schema,
% and SchemaIndexes a list of index/3 clauses for that schema.
% Creates a new index of form index(SchemaID, LH, RH) for each
% member of RHs.

index_one([], _, _, []).

index_one([RH/_ | RestRHs], ID, LH, [index(ID, LH, RH) | RestIndexes]) :-
    index_one(RestRHs, ID, LH, RestIndexes).

```



```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% collect_nodes/2
% arg1 = Leaves = list of all leaf nodes in all schemas, in
% list of form [Node1, ..., NodeN]; NB a leaf is a node which
% doesn't appear as a schema antecedent arg2 = Others = list of
% all non-leaf nodes in all schemas

collect_nodes(Roots, Leaves, Others) :-
    findall(Node,
        (index(_, Node, _); index(_, _, Node)),
        AllNodes),
    split(AllNodes, Roots, Leaves, Others).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% split(AllNodes, Roots, Leaves, Others)
% arg1 = list of all nodes (not including duplicates)
% arg2 = list of root nodes
% arg3 = list of all leaf nodes
% arg4 = list of all nodes which are not leaf or root nodes
% cuts prevent backtracking and reassignment of nodes to wrong lists

% 1 - all nodes processed
split([], [], [], []).

% 2 - when the node doesn't occur as a schema head, it is a leaf node
% and can immediately be assigned to Leaves
split([Node | RestNodes], Roots, [Node | RestLeaves], Others) :-
    \+ index(_, Node, _), !,
    split(RestNodes, Roots, RestLeaves, Others).

% 3 - when a node occurs as a schema head and in any consequent, assign
% it to Others
split([Node | RestNodes], Roots, Leaves, [Node | RestOthers]):-
    index(_, Node, _),
    index(_, _, Node), !,
    split(RestNodes, Roots, Leaves, RestOthers).

% 4 - otherwise, assign to Roots
split([Node | RestNodes], [Node | RestRoots], Leaves, Others) :-
    split(RestNodes, RestRoots, Leaves, Others).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PREDICATES FOR INFORMATIVITIES %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% with_informativities(Leaves, Others, NodesList)
% Leaf nodes are assigned an informativity of 1;
% Others have an informativity = sum of child node informativities + 0.5
% NodesList is a list of form [node(Node1, Inf1), ..., node(NodeN, InfN)]

with_informativities(Leaves, Others, NodesList) :-
    leaf_informativities(Leaves, LeafList),
    other_informativities(Others, LeafList, NodesList).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% leaf_informativities/2
% arg1 = Leaves = list of leaf nodes
% arg2 = NodesList = list of nodes of form node(Content, Informativity)

% 1 - terminating: all leaves assigned their informativity
leaf_informativities([], []).

% 2 - recursive: assign an informativity of 1 to next leaf node,
leaf_informativities([Node | T], [node(Node, 1) | NewT]) :-
    leaf_informativities(T, NewT).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% other_informativities/3
% arg1 = list of all non-leaf nodes, not annotated with informativities
% arg2 = Sofar = list of nodes so far annotated with informativity
% arg3 = NodesList = list of nodes of form node(Content, Informativity)
% NB when first called, the list of leaves is used as the sofar list

% 1 - terminating
other_informativities([], Sofar, Sofar).

% 2 - recursive: process the next element, then recurse
other_informativities([Node | T], Sofar, NodesList) :-
    node_informativity(Node, Sofar, NewSofar, _),
    other_informativities(T, NewSofar, NodesList).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% node_informativity(Node, NodesSofar, NewSofar, Informativity)
% arg1 = a node to be processed, so far not assigned informativity
% arg2 = list of nodes processed so far, in format
% [node(N1, Inf1), ..., node(Nm, Infm)]
% arg3 = used to update the sofar list, depending on whether Node
% has already been processed: if it has, NewSofar is the same as Sofar;
% if not, then Sofar is passed to sum_inf_rhs and may be updated from there
% arg4 = informativity of Node

% if Node has already been assigned informativity, then Sofar
% doesn't change; if not assigned, then find the sum of its right-hand
% sides and add 1; the new Sofar list is updated with the informativity
% of Node, plus any other definitions found along the way

node_informativity(Node, Sofar, NewSofar, Informativity) :-
    member(node(Node, Informativity), Sofar)
    ->                                     % if
    (NewSofar = Sofar);                    % then
    (findall(RH,                             % else
        index(_, Node, RH),
        RHs),
        sum_inf_rhs(RHs, Sofar, Sofar2, InfRHs),
        Informativity is InfRHs + 1,
        NewSofar = [node(Node, Informativity) | Sofar2])).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sum_inf_rhs(RHs, NodesSofar, NewNodesSofar, InfRHs)
% sums the informativity of a list of right-hand side nodes
% arg1 = list of all possible right-hands of a node
% arg2 = nodes so far assigned informativity
% arg3 = new list of nodes so far assigned
% arg4 = total informativity of all right-hand side nodes
% this calls sum_inf_rhs/5 - extra argument is an accumulator for keeping
% track of informativity so far

sum_inf_rhs(RHs, Sofar, NewSofar, InfRHs) :-
    sum_inf_rhs(RHs, Sofar, NewSofar, 0, InfRHs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sum_inf_rhs(RHs, NodesSofar, NewNodesSofar, Accumulator, InfRHs)

```

```
% find the informativity of the first consequent, add to accumulator,  
% then recurse  
% NB this calls node_informativity, which may update the Sofar list!  
% These two procedures are embedded in each other: difficult to  
% describe in English
```

```
sum_inf_rhs([], Sofar, Sofar, InfrHs, InfrHs).
```

```
sum_inf_rhs([Node | T], Sofar, NewSofar2, Acc, InfrHs) :-  
  node_informativity(Node, Sofar, NewSofar, Informativity),  
  NewAcc is Acc + Informativity,  
  sum_inf_rhs(T, NewSofar, NewSofar2, NewAcc, InfrHs).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%% TOTAL INFORMATIVITY OF ALL NODES %%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% total_inf/3  
% determines the total informativity of all nodes  
% arg1 = list of nodes of form node(Content, Inf)  
% arg2 = accumulator  
% arg3 = total informativity of all nodes
```

```
% 1 - terminating: informativity of all nodes has been summed  
total_inf([], Sum, Sum).
```

```
% 2 - recursive: find the basic_i value of each consequent,  
% then call recursively  
total_inf([node(_, Inf, _) | T], Acc, Sum) :-  
  NewAcc is Acc + Inf,  
  total_inf(T, NewAcc, Sum).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%%%%%%%%%%% PREDICATES FOR UBIQUITY %%%%%%%%%  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
% with_ubiquities/2  
% arg1 = NodeList1 = list of nodes of form node(NodeID, Informativity)  
% arg2 = NodeList2 = the new list of nodes of form
```

```

% node(NodeID, Informativity, Ubiquity)
% NB NodesList1 should not have any duplicates, as this is checked
% when assigning informativity

with_ubiquities(NodesWithInf, NodesWithInfAndUbi) :-
    with_ubiquities(NodesWithInf, [], NodesWithInfAndUbi).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% with_ubiquities/3
% Each node is assigned a ubiquity rating, equal to
% sum of ubiquities of parents + 1
% arg1 = NodeList1 = list of nodes of form node(NodeID, Informativity)
% arg2 = accumulated list of node definitions of form
% node(NodeID, Informativity, Ubiquity)
% arg3 = NodeList2 = the new list of nodes of form
% node(NodeID, Informativity, Ubiquity)

with_ubiquities([], Sofar, Sofar).

with_ubiquities([node(Node, Inf) | RestWithInf], Sofar,
NodesWithInfAndUbi) :-
    node_ubiquity(node(Node, Inf), Sofar, NewSofar, _),
    with_ubiquities(RestWithInf, NewSofar, NodesWithInfAndUbi).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% node_ubiquity(Node, NodesSofar, NewSofar, Informativity)
% arg1 = a node to be processed, so far not assigned ubiquity
% arg2 = list of nodes processed so far, in format
% [node(N1, Inf1), ..., node(Nm, Infm)]
% arg3 = used to update the sofar list, depending on whether Node
% has already been processed: if it has, NewSofar is the same as Sofar;
% if not, then Sofar is passed to sum_ubi_parents and may be updated
% from there
% arg4 = ubiquity of Node (NB this is used by sum_ubi_parents)

% if Node has already been assigned ubiquity, Sofar isn't updated;
% else, find the ubiquity of all parents (updating Sofar simultaneously);
% NewSofar has the new node definition as a head and the updated Sofar
% as a tail

node_ubiquity(node(Node, Inf), Sofar, NewSofar, Ubiquity) :-

```

```

member(node(Node, [Inf, Ubiquity]), Sofar)      % if
->
(NewSofar = Sofar)                             % then
;
(findall(Parent,                               % else
        index(_, Parent, Node),
        Parents),
 sum_ubi_parents(Parents, Sofar, Sofar2, UbiParents),
 Ubiquity is UbiParents + 1,
 NewSofar = [node(Node, [Inf, Ubiquity]) | Sofar2]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sum_ubi_parents(Parents, NodesSofar, NewNodesSofar, UbiParents)
% sums the ubiquity of a list of parents
% arg1 = list of parents of a node
% arg2 = nodes so far assigned ubiquity
% arg3 = new list of nodes so far assigned
% arg4 = total ubiquity of all parents
% this calls sum_ubi_parents/5 - extra argument is an accumulator
% for keeping track of ubiquity so far

sum_ubi_parents(Parents, Sofar, NewSofar, UbiParents) :-
    sum_ubi_parents(Parents, Sofar, NewSofar, 0, UbiParents).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sum_ubi_parents(Parents, NodesSofar, NewNodesSofar, Accumulator,
% UbiParents)
% find the ubiquity of the first parent, add to accumulator, then recurse
% NB this calls node_ubiquity, which may update the Sofar list!
% These two procedures are embedded in each other: difficult to
% describe in English

% 1 - terminating: all parents processed
sum_ubi_parents([], Sofar, Sofar, UbiParents, UbiParents).

% 2 - recursive: Ubi = sum Ubi(parents) + 1
sum_ubi_parents([Node | T], Sofar, NewSofar2, Acc, UbiParents) :-
    node_ubiquity(node(Node, _), Sofar, NewSofar, Ubiquity),
    NewAcc is Acc + Ubiquity,
    sum_ubi_parents(T, NewSofar, NewSofar2, NewAcc, UbiParents).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% flat(NodesList1, NodesList2)
% For matching purposes, NodesList1 has node definitions of the form
% node(Node, [Inf, Ubi])
% This procedure flattens these definitions into node/3 clauses of form
% node(Node, Inf, Ubi)

flat([], []).

flat([node(Node, [Inf, Ubi]) | RestNodes],
[node(Node, Inf, Ubi) | RestFlat]) :-
    flat(RestNodes, RestFlat).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% RUN ALL PROCEDURES WHICH CREATE FILES USED BY idc.pl
:- create_kb.

```


C.3 *IDC.pl*

```

/*****

```

```

IDC

```

```

(Incoherence-Driven Comprehender)

```

```

FILENAME:

```

```

['~/IDC_dir/idc.pl'].

```

```

FILES USED:

```

```

lists.pl

```

```

kb.pl

```

```

texts.pl

```

```

DATA STRUCTURES:

```

A representation has the form:

```

    repr(STS, LTS, STSInc)

```

STS = short-term store

LTS = long-term store

Memory stores have the form

```

    [Instance1, ..., InstanceN] ^ [Tree1, ..., TreeM]

```

where the first argument represents the instances, and the second the forest (list of trees).

Each instance has the form

```

    instance(Content, Parents, Children, Inc)

```

Parents (Par) is y if an instance occurs as a child in any tree (i.e. it has parents);

Children (Chd) is y if an instance occurs as a parent of any tree (i.e. it has children).

Content consists of raw content and a status marker

(# = inferred, @ = observed)

There are four types of instance content:

1. event - something which begins and ends within the time period of the story.
2. habit - something which is constant throughout the story (e.g. types, habitual propositions).
3. goal - mental events.
4. relation - describes a relationship between two or more instances.

There are a limited number of relation types, but unlimited habits, events and goals (i.e. there are no primitives here).

Where the ---> is placed between an event E and its subevents S (for example) this is taken to mean 'E explains S', where 'explains' has the broad meaning 'conceptually encapsulates or connects'. So, a relation can be used to explain the co-occurrence of two event instances.

Each tree has the form

```
tree(ID, Parent, Children, Inc)
```

where ID is the schema ID used in constructing the tree, Parent the parent instance of the tree, and Children the child instances list of the tree.

Instances and trees are collectively known as representational elements (r-elts). An r-elt only occurs in LTS or STS; never both.

Extensions - an extension has the structure:

```
ext(Repr, IncChange)
```

where Repr is a representation, and IncChange a term in one of the following forms:

```
obs(Message, IncChange)
    % a string describing how observation was
    % incorporated into representation; one of:
    % 'by merging with STS'
    % 'by merging with LTS'
    % 'by creation of a new instance'
```

```

infer(Method, Created, Removed)
    % Method = top-down, bottom-up, Created =
    % incchange due to tree
    % creation, Removed = incchange due to tree removal
transfer(IncChange)

*****/

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ADMINISTRATION PROCEDURES %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% quick reload
rl :- ['~/IDC_dir/idc.pl'].

% declare operator for inferred instances
:- op(100, fy, #).

% declare operator for observed instances
:- op(100, fy, @).

% declare operator for schema construction
% (so can load schemas.pl, with texts in it)
:- op(600, xfx, --->).

% operator for schema constraints
:- op(700, xfx, ==>).

% hack to allow direct output to a file
:- dynamic(output_mode/1).
output_mode(screen).

% SHARED PROCEDURES
% (variable binding, constraint checking, output, lists)
:- ['~/IDC_dir/shared.pl'].

% FILE CONTAINING INDICES, SCHEMAS AND NODE DEFINITIONS

```



```

% Tolerance equals (AvgInf - 1) * ToleranceFactor

comp(TextNum, Skept, Width, ToleranceFactor) :-
    text(TextNum, Text),
    avg_inf(AvgInf),
    Tolerance is ToleranceFactor * (AvgInf - 1),
    comprehend(Text, [repr([]^[], []^[], 0)], Skept, Tolerance,
                Width, 1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% filecomp(Filename, TextNum, Skept, Width, ToleranceFactor)
% As comp/4, but outputs to a text file rather than screen.
% Filename is an atom; the procedure creates an absolute file path
% in the IDC_dir directory, to which output is written.

filecomp(Filename, TextNum, Skept, Width, ToleranceFactor) :-
    retractall(output_mode(_)),
    assert(output_mode(file)),
    text(TextNum, Text),
    avg_inf(AvgInf),
    Tolerance is ToleranceFactor * (AvgInf - 1),
    name('~/IDC_dir/Output_dir/', L1),
    name(Filename, L2),
    name('.idc', L3),
    append(L1, L2, L4),
    append(L4, L3, L5),
    name(Outfile, L5),
    open(Outfile, write, Output1),
    set_output(Output1),
    write('FILENAME = '), write(Outfile), nl,
    write('TEXT = '), nl, write(Text), nl, nl,
    write('SKEPTICISM = '), write(Skept), nl,
    write('BEAM WIDTH = '), write(Width), nl,
    write('TOLERANCE FACTOR = '), write(ToleranceFactor),
    write('    TOLERANCE = '), write(Tolerance), nl, nl,
    write_line, nl, nl,
    comprehend(Text, [repr([]^[], []^[], 0)], Skept, Tolerance,
                Width, 1),
    retractall(output_mode(_)),
    assert(output_mode(screen)),
    close(Output1).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% comprehend(Text, CurrentReprs, Skept, Tolerance, BeamWidth, Cycle)
% CurrentReprs is sorted in ascending incoherence order, so the first
% representation should have the lowest incoherence (or equal lowest).

% 1 - recursive: text is not empty and incoherence is =< tolerance.
% OBSERVE NEXT STATEMENT
comprehend([Obs | RestObs], [Repr | RestCurrentReprs], Skept, Tolerance,
Range, Cycle1) :-
    inc_too_high(Repr, Tolerance, no),
    write_status(Cycle1, Skept, Tolerance, Range),
    write('Incorporating observation'), nl,
    tab(5), write(Obs), nl,
    write('into current representations'), nl,
    include_obs([Repr | RestCurrentReprs], Obs, Skept, NewExts),
    sort_exts(NewExts, [BestExt | RestSorted]),
    strip_exts([BestExt | RestSorted], NewReprs),
    storage(NewReprs, Load),
    write_storage(Load),
    write_exts([BestExt | RestSorted]),
    Cycle2 is Cycle1 + 1,
    comprehend(RestObs, NewReprs, Skept, Tolerance, Range, Cycle2).

% 2 - recursive: Text can be empty or full.
% N.B. this also prunes the number of representations, according to Range
% and will only be called if at least one representation contains instances
% in STS
% This clause fails if no extensions are returned by action/4
% CREATE AND CHECK
comprehend(Text, [Repr | RestCurrentReprs], Skept, Tolerance, Range, Cycle1) :-
    member(repr([_ | _] ^ _, _, _), [Repr | RestCurrentReprs]),
    action([Repr | RestCurrentReprs], create_and_check, Skept, NewExts),
    sort_and_trim_exts(Range, NewExts, [BestExt | RestSorted]),
    write_status(Cycle1, Skept, Tolerance, Range),
    write('Incoherence in STS too high, so creating new inferences'),
    nl,
    strip_exts([BestExt | RestSorted], NewReprs),
    storage(NewReprs, Load),
    write_storage(Load),
    write_exts([BestExt | RestSorted]),

```

```

    Cycle2 is Cycle1 + 1,
    comprehend(Text, NewReprs, Skept, Tolerance, Range, Cycle2).

% 3 - recursive: Text can be empty or full, but there must be
% something in the first representation
% TRANSFER
comprehend(Text, [Repr | RestReprs], Skept, Tolerance, Range, Cycle1) :-
\+ Repr = repr([], [], _, _),
inc_too_high(Repr, Tolerance, yes),
clear_STS([Repr | RestReprs], Skept, Tolerance, Range, Cycle1,
NewReprs, Cycle2),
comprehend(Text, NewReprs, Skept, Tolerance, Range, Cycle2), !.

% 4 - terminating: text is empty and no further operations can lower
% incoherence
comprehend([], Reprs, _, _, _, _) :-
write('TEXT IS FULLY PROCESSED'), nl,
write('COHERENCE VALUES BELOW ARE ABSOLUTE, WRT SKEPTICISM =
0.5'), nl, nl,
write_final_reprs(Reprs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% clear_STS(Text, NewReprs, Skept, Tolerance, Range, Cycle, NewReprs2,
% Cycle2)
% Remove elements from STS until incoherence falls below Tolerance

% 1 - terminating: return if incoherence of first representation is
% below tolerance
clear_STS([Repr | RestReprs], _, Tolerance, _, Cycle, [Repr | RestReprs],
Cycle) :-
inc_too_high(Repr, Tolerance, no), !.

% 2 - recursive: do a transfer, then call recursively
clear_STS(Reprs, Skept, Tolerance, Range, Cycle1, NewReprs2, Cycle3) :-
do_transfer(Reprs, Skept, NewExts),
sort_exts(NewExts, [BestExt | RestSorted]),
write_status(Cycle1, Skept, Tolerance, Range),
write('Transferring elements from current representations
STS to LTS'),
nl,
strip_exts([BestExt | RestSorted], NewReprs),

```

```

storage(NewReprs, Skept, Load),
write_storage(Load),
write_exts([BestExt | RestSorted]),
Cycle2 is Cycle1 + 1,
clear_STS(NewReprs, Skept, Tolerance, Range, Cycle2, NewReprs2,
Cycle3).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% inc_too_high(Repr, Tolerance, Answer)
% Returns 'no' if the incoherence of Repr =< Tolerance; otherwise, 'yes'
% (Saves some ugly code in comprehend)

inc_too_high(repr(_, _, Inc), Tolerance, Answer) :-
    Inc =< Tolerance
    ->
    Answer = no
    ;
    Answer = yes.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% strip_exts(Extensions, Reprs)
% Strips the rating off each extension in Extensions to leave the
% corresponding representations Reprs

strip_exts([], []).

strip_exts([ext(Repr, _) | RestExts], [Repr | RestReprs]) :-
    strip_exts(RestExts, RestReprs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% storage(Reprs, Load)
% Load = fixed incoherence of unique instances and trees in Reprs
% N.B. it uses a copy of current representations, to prevent
% untoward unification

storage(Reprs, Load) :-
    copy_term(Reprs, ReprsCopy),

```



```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% WRITING FINAL REPRESENTATIONS %%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_final_reprs(Reprs)
% For each representation in Reprs, determines the fixed incoherence
% (wrt a Skepticism of 0.5) of the representation and writes it to the
% screen

% 1 - terminating: all final representations written
write_final_reprs([]) :-
    write('ALL REPRESENTATIONS WRITTEN'), nl, nl.

% 2 - recursive: determine the fixed incoherence of the next
% representation and write to screen, then recurse
write_final_reprs([repr(STS, LTS, _) | RestReprs]) :-
    fixed_incoherence(STS, STSInc),
    fixed_incoherence(LTS, LTSInc),
    FixedInc is STSInc + LTSInc,
    write_repr(repr(STS, LTS, FixedInc)),
    coherence(FixedInc, Cohr),
    write('Coherence is '), write(Cohr), nl, nl,
    continue(repr(STS, LTS, FixedInc)),
    write_final_reprs(RestReprs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% fixed_incoherence(Exgraph, FixedInc)
% Evaluates an exgraph with respect to a skepticism of 0.5;
% this gives the 'absolute' value of a exgraph

fixed_incoherence(Instances ^ Forest, FixedInc) :-
    fixed_instances_inc(Instances, 0, FixedInstancesInc),
    fixed_trees_inc(Forest, 0, FixedForestInc),
    FixedInc is FixedInstancesInc + FixedForestInc.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% fixed_instances_inc(Instances, Acc, FixedInstancesInc)

fixed_instances_inc([], Acc, Acc).

fixed_instances_inc([instance(Content, Par, Chd, _) | RestInstances],

```

```

Acc, FixedInstancesInc) :-
    instance_inc(instance(Content, Par, Chd), 0.5, _, OneInc),
    NewAcc is Acc + OneInc,
    fixed_instances_inc(RestInstances, NewAcc, FixedInstancesInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% fixed_trees_inc(Forest, Acc, FixedTreesInc)

fixed_trees_inc([], Acc, Acc).

fixed_trees_inc([tree(ID, Parent, Children, _) | RestTrees], Acc,
FixedTreesInc) :-
    tree_inc(tree(ID, Parent, Children), 0.5, _, OneInc),
    NewAcc is Acc + OneInc,
    fixed_trees_inc(RestTrees, NewAcc, FixedTreesInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% coherence(FixedInc, Cohr)

coherence(FixedInc, Cohr) :-
    total_inf(TotInf),
    findall(N, node(N, _, _), Nodes),
    length(Nodes, TotalNoNodes),
    MaxInc is ((TotInf - TotalNoNodes) * 2),
    IncUsed is MaxInc - FixedInc,
    Cohr is IncUsed / MaxInc.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% GENERATING NEW EXTENSIONS %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% action(CurrentReprs, ActionType, Skept, NewReprs)
% Carries out ActionType on all reprs in CurrentReprs in turn.

% 1 - terminating
action([], _, _, []).

```

```

% 2 - recursive: find extensions of first representation, then recurse.
action([Repr | RestReprs], Type, Skept, NewReprs) :-
    action_narrative(Type, Repr, Skept, Goal, Output),
    findall(Output,
        Goal,
        OneNewGen),
    action(RestReprs, Type, Skept, RestNewGen),
    append(OneNewGen, RestNewGen, NewReprs).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% action_narrative(ActionType, ReprIn, Skept, Goal, ReprOut)
% ActionType =
%     create_and_check (connect instances then remove redundancies
%     and spuriousness in a single cycle)
%     transfer_tree (move tree and dependent instances from STS to LTS)
%     transfer_instance (move isolated instance from STS to LTS)
% Each action has the same input but a different set of procedure
% calls in Goal.

% 1 - create_and_check: extend the representation, then check whether the
% new tree used to extend the representation has made any others redundant.
% For reprs with at least one instance in STS
action_narrative(create_and_check, repr(STS1, LTS1, Inc), Skept,
    ((STS1 = [_ | _] ^ _), create_and_check(repr(STS1, LTS1, Inc),
    Skept, repr(STS2, LTS2, NewInc), infer(Method, Created, Removed)),
    Incchange is Created + Removed, Incchange < 0),
    ext(repr(STS2, LTS2, NewInc), infer(Method, Created, Removed))).

% 2 - create_and_check for empty STS
action_narrative(create_and_check, repr(STS, LTS, Inc), _,
    (STS = [] ^ _),
    ext(repr(STS, LTS, Inc), infer('no inference', 0, 0))).

% 3 - transfer_tree: transfer a tree to LTS, along with its
% dependent instances.
% IncChange for a transfer = NewInc - InitialInc.
% This clause works when STS actually has something in it
action_narrative(transfer_tree, repr(STS, LTS, Inc), Skept,
    (\+(STS = _ ^ []), transfer_tree(repr(STS, LTS, Inc),
    Skept, repr(NewSTS, NewLTS, NewInc))), IncChange is NewInc - Inc,

```

```

    IncChange =< 0),
    ext(repr(NewSTS, NewLTS, NewInc), transfer(IncChange))).

% 4 - transfer_tree for representations with empty STS forest
action_narrative(transfer_tree, repr(STS, LTS, Inc), _,
    (STS = _ ^ []),
    ext(repr(STS, LTS, Inc), transfer(0))).

% 5 - transfer_instance: transfer a parent- and child-less instance to LTS.
% For representations containing at least one instance
action_narrative(transfer_instance, repr(STS, LTS, Inc), _,
    (\+ (STS = [] ^ _), transfer_instance(repr(STS, LTS, Inc),
    repr(NewSTS, NewLTS, NewInc)), IncChange is NewInc - Inc,
    IncChange =< 0),
    ext(repr(NewSTS, NewLTS, NewInc), transfer(IncChange))).

% 6 - transfer_instance for representations with empty STS instances
action_narrative(transfer_instance, repr(STS, LTS, Inc), _,
    (STS = [] ^ _),
    ext(repr(STS, LTS, Inc), transfer(0))).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SORT AND TRIM %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sort_and_trim_exts(Range, Extensions, NewExts)
% NewExts consists of the best extension in Extensions1, plus other
% extensions whose inc is within Range * Inc of the best representation.
% The larger range is, the more extensions are acceptable.
% If Range = 1, all representations with the same incoherence are maintained.
% providing they consist of different trees.

% 1 - if Range = -1, only a single best extension is accepted
sort_and_trim_exts(-1, [ext(repr(STS1, LTS1, Inc), Rating) | RestExtensions],
[BestExt]) :-
    find_best(RestExtensions, ext(repr(STS1, LTS1, Inc), Rating),

```



```

% If first extension in Extensions has higher inc than the second,
% swap them over; otherwise, leave first where it is, and move onto second.

% 1 - first extension has lower inc than second
swap_ext([ext(repr(STS1, LTS1, Inc1), Rating1),
          ext(repr(STS2, LTS2, Inc2), Rating2) | RestExts],
         [ext(repr(STS2, LTS2, Inc2), Rating2),
          ext(repr(STS1, LTS1, Inc1), Rating1) | RestExts)]) :-
    Inc2 < Inc1.

% 2 - leave first two elements in same order
swap_ext([H | T1], [H | T2]) :-
    swap_ext(T1, T2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% range_trim(Range, ListIn, ListOut)
% Use the first element of ListIn as the 'standard'; each other
% representation whose inc is above the standard consumes part of
% that resource (so range acts like an 'activation pool' and thus
% more like 'capacity')

range_trim(Range, [ext(repr(STS1, LTS1, Inc1), Rating) | RestIn], ListOut) :-
    range_trim(RestIn, Inc1, Range, 0,
               [ext(repr(STS1, LTS1, Inc1), Rating)], ListOut).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% range_trim(ExtsIn, LowestInc, Range, EffortAcc, ExtsSofar, ExtsOut)
% LowestInc is the incoherence of the best repr found so far

% 1 - terminating: no more extensions to check, so set outputs to
% accumulators
range_trim([], _, _, _, ExtsOut, ExtsOut).

% 2 - check that next extension has a valid rating and won't
% exceed Range:
% if it doesn't, all usable exts have been checked, so call
% with empty list to terminate; if it does, call attempt_append,
% which tries to add the extension to the sofar list
range_trim([ext(repr(STS, LTS, Inc), Rating) | RestExts],
LowestInc, Range, Effort1, Sofar, ExtsOut) :-
    EffortForExt is (Inc - LowestInc),

```

```

clip(EffortForExt, EffortForExt2),
Effort2 is Effort1 + EffortForExt2,
Effort2 =< Range
->
attempt_append(ext(repr(STS, LTS, Inc), Rating),
               EffortForExt, Sofar, ActualEffort, NewSofar),
Effort3 is Effort1 + ActualEffort,
range_trim(RestExts, LowestInc, Range, Effort3, NewSofar, ExtsOut)
;
range_trim([], _, _, _, Sofar, ExtsOut).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% attempt_append(Extension, EffortForExt, Sofar, ActualEffort, NewSofar)
% If Extension in not duplicated in Sofar, add it to Sofar and set
% Effort3 to EffortForExt; else, NewSofar = Sofar and ActualEffort = 0

attempt_append(Extension, EffortForExt, Sofar, ActualEffort, NewSofar) :-
    \+ duplicate_ext(Sofar, Extension)
->
    ActualEffort = EffortForExt,
    append(Sofar, [Extension], NewSofar)
;
    ActualEffort = 0,
    NewSofar = Sofar.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% duplicate_ext(OtherExtensions1, Extension)
% Extension is duplicated in OtherExtensions if its
% incoherence, STSF and LTSF match the Inc, STSF and LTSF of some extension
% in OtherExtensions.

% 1 - next extension is a duplicate of Extension as forests contain
% identical trees, not necessarily occurring in the same order.
duplicate_ext([ext(repr(_ ^ STSF1, _ ^ LTSF1, Inc), _) | _],
              ext(repr(_ ^ STSF2, _ ^ LTSF2, Inc), _)) :-
    identical_forests(STSF1, STSF2),
    identical_forests(LTSF1, LTSF2), !.

% 2 - recursive: next extension is not a duplicate, try elsewhere in list
duplicate_ext([_ | T], Extension) :-
    duplicate_ext(T, Extension).

```



```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% integrate_obs_with_repr(Repr, Obs, Skept, NewExt)
% Repr = repr(STSExgraph, LTSExgraph, STSInc)
% Obs = a new observation, without any status token (no '@')
% NewExt = Repr with Obs integrated into it, plus message and incchange
% A new observation may be merged with an existing instance in
% STS or LTS, or added to STS or LTS without merging.
% If merged with an existing instance, updating of trees in STS and/or LTS is
% required (i.e. changing inferred instances to observed instances).
% Wherever structures are altered by the new observation, incoherence of
% those structures is updated; STSInc is also updated if STS is
% changed in any way.

% 1 - merge observation with an existing STS instance, and update trees.
integrate_obs_with_repr(repr(I1a ^ F1a, I2a ^ F2a, Inc), Obs, Skept,
ext(repr(I1c ^ F1c, I2a ^ F2b, NewInc), obs('merging with STS', IncChange))) :-
    merge_obs_with_instance_list(I1a, Obs, Skept, NewInstance, I1b),
    update_forest(F1a, NewInstance, Skept, F1b),
    update_forest(F2a, NewInstance, Skept, F2b),
    inc([NewInstance | I1b] ^ F1b, Skept, I1c ^ F1c, NewInc),
    IncChange is NewInc - Inc, !.

% 2 - merge observation with an existing LTS instance, and update trees.
% Note that the LTS instance is considered to have been retrieved into STS -
% this allows new inferences based on the new observation to be made.
integrate_obs_with_repr(repr(I1a ^ F1a, I2a ^ F2a, Inc), Obs, Skept,
ext(repr(I1b ^ F1c, I2b ^ F2b, NewInc), obs('merging with LTS', IncChange))) :-
    merge_obs_with_instance_list(I2a, Obs, Skept, NewInstance, I2b),
    update_forest(F1a, NewInstance, Skept, F1b),
    update_forest(F2a, NewInstance, Skept, F2b),
    inc([NewInstance | I1a] ^ F1b, Skept, I1b ^ F1c, NewInc),
    IncChange is Inc - NewInc, !.

% 3 - add observation to STS instances without updating any r-elts.
integrate_obs_with_repr(repr(I1 ^ F1, I2 ^ F2, Inc), Obs, Skept,
ext(repr([NewInstance | I1] ^ F1, I2 ^ F2, NewInc),
obs('creating new instance', InstanceInc))) :-
    instance_inc(instance(@Obs, n, n), Skept, NewInstance, InstanceInc),
    NewInc is Inc + InstanceInc.

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% MERGING AN OBSERVATION WITH A INSTANCE %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% merge_obs_with_instance_list(CurrentInstances, NewObs, Skept,
% NewInstance, CurrentInstancesLeft)
% Merges NewObs with an element of CurrentInstances to leave
% CurrentInstancesLeft; NewInstance is the instance created from
% NewObs (e.g. if an existing inferred instance has instantiated
% arguments which unify with arguments of NewObs).
% This is used to update the trees in STS and LTS.
% NewObs is not marked as an observation with '@' at this point.

% 1 - terminating: next elt can be merged with the next instance
% in the instance list.
merge_obs_with_instance_list([Instance1 | RestInstances1], NewObs,
Skept, instance(@Content, Par, Chd, Inc), RestInstances1) :-
    combine_obs_with_instance(NewObs, Instance1, Skept,
        instance(@Content, Par, Chd, Inc)), !.

% 2 - recursive: try next element of Instances1.
merge_obs_with_instance_list([Instance1 | RestInstances1], NewObs,
Skept, NewInstance, [Instance1 | RestInstances2]) :-
    merge_obs_with_instance_list(RestInstances1, NewObs, Skept,
        NewInstance, RestInstances2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% combine_obs_with_instance(NewObs, ExistingInstance, Skept, NewInstance)
% Combines a new obs with an existing instance (which may have unbound
% event tokens). This should bind the values in the new instance to
% unbound values in the existing instance. NewContent is the content
% of a new observation, while ExistingInstance has is an instance with
% inc assigned.
% NB a new observation cannot be combined with a previous observation.

% 1 - for all except relations
combine_obs_with_instance(C1, instance(C2, Par2, Chd2, _),
Skept, NewInstance) :-
    \+ C2 = @_,
    \+ C1 = relation(_, _),

```



```
% update_tree(NewInstance, Tree, Skept, NewTree)
% Updates any instance in Tree whose content exactly matches the
% content of NewInstance. Inc is calculated for the updated tree.
% NB the checks on flag content ensure that the correct clause is called.
```

```
% 1 - update a tree's parent.
update_tree(instance(@Content, _, y, _), tree(ID, #Parent, Children, _),
Skept, NewTree) :-
    Content == Parent,
    tree_inc(tree(ID, @Content, Children), Skept, NewTree, _).
```

```
% 2 - update a tree's child.
update_tree(instance(@Content, y, _, _), tree(ID, Parent, Children1, _),
Skept, NewTree) :-
    indexed(ID, _, @Content),
    update_children(Content, Children1, Children2),
    tree_inc(tree(ID, Parent, Children2), Skept, NewTree, _).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% update_children(Content, Children1, Children2)
% Updates the status of any child whose content matches Content
% (which is the content of an observed instance).
```

```
update_children(C1, [#C2 | Rest], [@C2 | Rest]) :-
    C1 == C2, !.
```

```
update_children(C1, [C2 | Rest2], [C2 | Rest3]) :-
    \+ loose_match(@C1, C2),
    update_children(C1, Rest2, Rest3).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% CREATE_AND_CHECK PROCEDURES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create_and_check(Repr, Skept, NewRepr, IncChange)
% Extends a repr, then removes from that repr any trees and
```



```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create_top_down(Repr, Skept, NewRepr, IncChange)
% Finds a possible top-down extension NewRepr of Repr.
% NB CANNOT INFER DOWNWARDS FROM A RELATION
% Keeps track of the change to incoherence caused by making the
% extension - both in STS, in terms of new instances created and
% instances which gain parents or children.
% IncChange is the change in incoherence of STS elts: if positive,
% incoherence has been increased and coherence reduced; if negative,
% incoherence has been reduced by that amount.

create_top_down(repr(STSI1 ^ STSF1, LTSI1 ^ LTSF, _), Skept,
repr(NewSTS, LTSI2 ^ LTSF, NewInc), IncChange2) :-
    % select instance for parent
    select_instance(any, STSI1, instance(P, Par, _, NInc1), STSI2),
    raw_content(P, RawP),
    % prevent inferring down from relations
    \+ RawP = relation(_, _),
    ID: RawP/Cons ---> RHs,
    % prevent duplication
    \+ tree_present(tree(ID, P, _, _), STSF1),
    instance_inc(instance(P, Par, y), Skept, ParentInstance, NInc2),
    Acc is NInc2 - NInc1,
    construct_children(RHs, ID, STSI2, LTSI1, Skept, Acc,
        STSI3, LTSI2, Children, IncChange1),
    check(Cons),
    tree_inc(tree(ID, P, Children), Skept, NewTree, TreeInc),
    IncChange2 is IncChange1 + TreeInc,
    inc([ParentInstance | STSI3] ^ [NewTree | STSF1], Skept,
        NewSTS, NewInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
EXTEND_BOTTOM_UP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create_bottom_up(Repr, Skept, NewRepr, IncChange)
% Finds a possible bottom up extension NewRepr of Repr.
% NB CANNOT INFER UPWARDS FROM A RELATION

```

```

create_bottom_up(repr(STSI1 ^ STSF1, LTSI1 ^ LTSF, _), Skept,
repr(NewSTS, LTSI3 ^ LTSF, NewInc), IncChange4) :-
    % select instance for first child
    select_instance(any, STSI1, instance(C, _, Chd1, Inc1), STSI2),
    raw_content(C, RawC),
    % prevent inferring up from relations
    \+ RawC = relation(_, _),
    indexed(ID, #P, C),
    ID: P/Cons ---> RHs,
    % prevent duplication
    \+ tree_present(tree(ID, #P, _, _), STSF1),
    construct_parent(LTSI1, STSI2, P, ID, Skept, LTSI2, STSI3,
        Parent, IncChange1),
    construct_first_child(C, RHs, Children1),
    instance_inc(instance(C, y, Chd1), Skept, NewInstance, Inc2),
    IncChange2 is IncChange1 + (Inc2 - Inc1),
    construct_children(Children1, ID, STSI3, LTSI2, Skept,
        IncChange2, STSI4, LTSI3, Children2, IncChange3),
    check(Cons),
    tree_inc(tree(ID, Parent, Children2), Skept, NewTree, TreeInc),
    IncChange4 is IncChange3 + TreeInc,
    inc([NewInstance | STSI4] ^ [NewTree | STSF1], Skept,
        NewSTS, NewInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% TREE AND INSTANCE CONSTRUCTION %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% construct_parent(LTSI1, STSI1, RawContent, ID, Skept, LTSI2, STSI2,
% Parent, IncChange)
% ParentInstance is either constructed by directed unification with
% an instance in LTS, or by creation of a new instance.

% 1 - RawContent doesn't match any instance in LTSI1, so create a
% new instance (providing it isn't already present in STSI)
construct_parent([], STSI, C, _, Skept, [], [NewInstance | STSI],
#C, IncChange) :-
    \+ instance_duplicated(instance(#C, n, y, _), STSI),
    instance_inc(instance(#C, n, y), Skept, NewInstance, IncChange).

```



```

% 2 - RawContent matches the next instance in LTSN
construct_parent([instance(C2, Par, _, Inc1) | RestStore], STSI,
C1, ID, Skept, [NewInstance | RestStore], STSI, C2, IncChange) :-
    indexed(ID, C2, #_),
    raw_content(C2, RawC2),
    combine_raw_content(RawC2, C1),
    instance_inc(instance(C2, Par, y), Skept, NewInstance, Inc2),
    IncChange is Inc2 - Inc1, !.

% 3 - try later instance in LTSN
construct_parent([Instance | RestInstances1], STSI1, C, ID, Skept,
[Instance | RestInstances2], STSI2, Parent, IncChange) :-
    construct_parent(RestInstances1, STSI1, C, ID, Skept,
        RestInstances2, STSI2, Parent, IncChange).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% construct_first_child(Content, RHs, Children)
% Takes the trigger instance's content and inserts it into the
% correct place in the RHs list to give Children1; this ensures
% that the instance which triggered the bottom-up extension is
% included in the resulting tree.

% 1 - terminating: next RHs instance matches Content.
construct_first_child(C, [RH/Cons | RestRHs], [C | RestRHs]) :-
    raw_content(C, RawC),
    combine_raw_content(RawC, RH),
    check(Cons), !.

% 2 - recursive: check later RHs instances.
construct_first_child(C, [RH | RestRHs1], [RH | RestRHs2]) :-
    construct_first_child(C, RestRHs1, RestRHs2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% construct_children(Children1, ID, STSI1, LTSI1, Skept, Acc, STSI2,
% LTSI2, Children2, IncChange)
% Children1 = list of RH nodes of a schema and possibly one
% already-inferred child; RH nodes have form Content/Constraints.
% ID = ID of schema used to construct new tree, used to speed processing.

% 1 - terminating: no more RH nodes to match.

```

```
construct_children([], _, STSI, LTSI, _, IncChange, STSI, LTSI, [],
IncChange).
```

```
% 2 - recursive: next RH node can be matched with an instance in STS.
construct_children([C/Cons | RestChildren1], ID, STSI1, LTSI1, Skept,
Acc, STSI3, LTSI2, [Child | RestChildren2], IncChange) :-
    unify_RH_with_instance(C/Cons, ID, STSI1, Skept, STSI2, Child,
        SubIncChange),
    NewAcc is Acc + SubIncChange,
    construct_children(RestChildren1, ID, STSI2, LTSI1, Skept, NewAcc,
        STSI3, LTSI2, RestChildren2, IncChange), !.
```

```
% 3 - recursive: next RH node can be matched with an instance in LTS.
construct_children([C/Cons | RestChildren1], ID, STSI1, LTSI1, Skept, Acc,
STSI2, LTSI3, [Child | RestChildren2], IncChange) :-
    unify_RH_with_instance(C/Cons, ID, LTSI1, Skept, LTSI2, Child,
        SubIncChange),
    NewAcc is Acc + SubIncChange,
    construct_children(RestChildren1, ID, STSI1, LTSI2, Skept,
        NewAcc, STSI2, LTSI3, RestChildren2, IncChange), !.
```

```
% 4 - recursive: no match in STS or LTS, so infer new instance and add to STS.
construct_children([C/Cons | RestChildren1], ID, STSI1, LTSI1, Skept, Acc,
[NewInstance | STSI2], LTSI2, [#C | RestChildren2], IncChange) :-
    check(Cons),
    instance_inc(instance(#C, y, n), Skept, NewInstance,
        NewInstanceInc),
    NewAcc is Acc + NewInstanceInc,
    construct_children(RestChildren1, ID, STSI1, LTSI1, Skept,
        NewAcc, STSI2, LTSI2, RestChildren2, IncChange).
```

```
% 5 - recursive: next child has already been constructed
% (e.g. by construct_first_child)
construct_children([C | RestChildren1], ID, STSI1, LTSI1, Skept, Acc,
STSI2, LTSI2, [C | RestChildren2], IncChange) :-
    raw_content(C, _),
    construct_children(RestChildren1, ID, STSI1, LTSI1, Skept,
        Acc, STSI2, LTSI2, RestChildren2, IncChange).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% unify_RH_with_instance(RH, ID, CurrentInstances, Skept, NewInstances,
```

```

% Child, IncChange)
% RH = a right-hand node, complete with its constraints list.
% ID = ID of the schema being used to construct the tree.
% CurrentInstances = list of instances current in a store.
% NewInstances = updated list of instances.
% Child = a new child for a tree, based on the unification of RH and an
% instance in the store.
% IncChange = change to the incoherence of the instance which was matched.

% 1 - terminating: InstanceContent matches next RH node and constraints hold.
unify_RH_with_instance(RH/Cons, ID,
[instance(N, _, Chd, Inc1) | RestInstances], Skept,
[NewInstance | RestInstances], N, ChildIncChange) :-
    indexed(ID, #_, N),
    raw_content(N, RawN),
    combine_raw_content(RawN, RH),
    check(Cons),
    instance_inc(instance(N, y, Chd), Skept, NewInstance, Inc2),
    ChildIncChange is Inc2 - Inc1, !.

% 2 - recursive: try a later instance in the list.
unify_RH_with_instance(RH/Cons, ID, [Instance | RestInstances1], Skept,
[Instance | RestInstances2], Child, ChildIncChange) :-
    unify_RH_with_instance(RH/Cons, ID, RestInstances1, Skept,
        RestInstances2, Child, ChildIncChange).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PROCEDURES FOR REMOVING REDUNDANT ELEMENTS %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove_redundancies(Tree, STS1, LTS1, Skept, STS2, LTS2, IncChange)
% Removes all redundant trees and associated instances from STS1 and
% LTS1 to give STS2 and LTS2.
% 1. Check each tree in STS first - if made redundant by Tree,
% remove from STS then call update_repr on the remainder of
% STS and current LTS.

```

```

% 2. Check each tree in LTS - do as 1 for each redundant tree.

remove_redundancies(Tree, STS1, LTS1, Skept, STS3, LTS3, IncChange2) :-
    remove_redundant_trees(Tree, STS1, LTS1, Skept, 0, STS2, LTS2,
        IncChange1),
    remove_redundant_trees(Tree, LTS2, STS2, Skept, IncChange1,
        LTS3, STS3, IncChange2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove_redundant_trees(Tree, Exgraph1a, Exgraph2a, Skept, Acc,
% Exgraph1b, Exgraph2b, IncChange)
% Checks each tree in Exgraph1a: if it is redundant wrt Tree,
% propagate the changes through both Exgraph1a and Exgraph2a to
% give new exgraphs which are passed to the recursive call; otherwise
% (the tree isn't redundant), make no changes.
% Acc keeps track of the changes to inc.

remove_redundant_trees(Tree, I1a ^ F1a, I2a ^ F2a, Skept, Acc,
I1b ^ F1b, I2b ^ F2a, IncChange) :-
    append(F1a, F2a, AllTrees),
    remove_redundant_trees1(F1a, Tree, AllTrees, I1a, I2a,
        Skept, Acc, I1b, F1b, I2b, IncChange).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove_redundant_trees1(F1a, Tree, AllTrees, I1a, I2a, Skept, Acc,
% I1b, F1b, I2b, IncChange)
% Each tree in F1a is checked in turn; if redundant with respect to Tree,
% update the values of the instances in I1a and I2a to give I1b and
% I2b respectively to take account of removal of this tree.
% If a tree is not redundant, it can just be added to the output.

% 1 - terminating: all trees checked.
remove_redundant_trees1([], _, _, I1, I2, _, Acc, I1, [], I2, Acc).

% 2 - recursive: if next tree is redundant, update stores to account
% for this; otherwise, add it to the output.
remove_redundant_trees1([Tree1 | RestTrees], Tree2, AllTrees, I1a,
I2a, Skept, Acc, I1c, F1, I2c, IncChange) :-
    redundant(Tree1, Tree2)
    ->
    remove(Tree1, AllTrees, AllTrees2),

```

```

update_stores(Tree1, [Tree2 | AllTrees2], I1a, I2a,
              Skept, I1b, I2b, SubIncChange),
NewAcc is Acc + SubIncChange,
remove_redundant_trees1(RestTrees, Tree2, AllTrees2, I1b,
                        I2b, Skept, NewAcc, I1c, F1, I2c, IncChange)
;
remove_redundant_trees1(RestTrees, Tree2, AllTrees, I1a,
                        I2a, Skept, Acc, I1c, F1T, I2c, IncChange),
F1 = [Tree1 | F1T].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% REDUNDANCY CHECKING %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% redundant(Tree1, Tree2)
% Checks whether Tree1's observed children are a variant subset of
% the observed children of the second tree; if this is the case,
% the first tree is redundant (i.e. the first tree explains the
% same or fewer observations as second tree). The first tree is required
% to have at least one observed child, otherwise it cannot be considered
% redundant with respect to a second tree.
% (It could be considered spurious, though.)

% 1 - Tree1 is redundant wrt Tree2 if Tree1's observed children
% are a variant of Tree2's and Tree2 has lower inc than Tree1;
% also, the parent of Tree1 must not be a child of Tree2, and
% the trees must not unify.
redundant(tree(_, P1, C1, Inc1), tree(_, P2, C2, Inc2)) :-
  \+ tree(_, P1, C1, Inc1) = tree(_, P2, C2, Inc2),
  % ensure that Tree1 has at least one observed child
  member(@_, C1),
  \+ strict_child_present(P1, C2),
  variant(C1, C2),
  Inc1 >= Inc2, !.

% 2 - Tree1 is redundant wrt Tree2 if their parents and IDs exactly
% match, but Tree2 has equal or lower inc, e.g. if the same instance
% is used to construct two trees and one has more accurate
% 'elaborations' (less incoherent).
redundant(tree(ID, P1, _, Inc1), tree(ID, P2, _, Inc2)) :-

```

```

P1 == P2,
Inc1 >= Inc2.

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% variant(Children1, Children2)
% True if the first tree's observed children are a subset or equal to
% the observed members of Children2.

```

```

% 1 - terminating: all observed elements of Children1 are present
% in Children2.
variant([], _).

```

```

% 2 - recursive: next child of the first tree is observed and
% strictly present in Children2.
variant([@Child1 | Rest1], Children2) :-
    strict_remove(@Child1, Children2, Children3),
    variant(Rest1, Children3).

```

```

% 3 - recursive: next child of the first tree is inferred, so ignored.
variant([#_ | Rest1], Children2) :-
    variant(Rest1, Children2).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PROCEDURES FOR REMOVING SPURIOUS ELEMENTS %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

% A potentially spurious tree is recognised by the fact that it
% contains no observations, either as its parent or among its
% observations; and its parent has no parent of its own (i.e.
% Parent flag set to 'n'); and its children have no children of their own.
% A spurious tree T1 generally comes about when a tree T2 is
% superceded by a new tree T3; if one of the children of T2 was
% inferred, and this child was the parent of T1, then T1 may
% become spurious (as its parent was dependent on a now-redundant tree).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove_spurious(Exgraph1, Exgraph2, Skept, NewExgraph1, NewExgraph2,

```

```

% IncChange)
% If a spurious tree can be found and removed, along with its
% associated r-elts, then recurse with the remainder of the representation;
% otherwise, just return the current STS and LTS;
% Acc is the IncChange so far generated by removing spurious trees.
% NB this is quite complicated, as it first tries removing the tree and
% its instances; if this produces an improvement, the tree can be
% removed from the list of trees to be processed - however, it still
% has to be in AllTrees, for the purposes of updating other instances
% which may be in potentially spurious trees.

remove_spurious(STS1, LTS1, Skept, STS3, LTS3, IncChange2) :-
    remove_spurious_trees(STS1, LTS1, Skept, 0, STS2, LTS2,
        IncChange1),
    remove_spurious_trees(LTS2, STS2, Skept, IncChange1, LTS3,
        STS3, IncChange2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove_spurious_trees(Exgraph1a, Exgraph2a, Skept, Acc, Exgraph1b,
% Exgraph2b, IncChange)

remove_spurious_trees(I1a ^ F1a, I2a ^ F2a, Skept, Acc, I1b ^ F1b,
I2b ^ F2a, IncChange) :-
    append(F1a, F2a, AllTrees),
    remove_spurious_trees1(F1a, AllTrees, I1a, I2a, Skept,
        Acc, I1b, F1b, I2b, IncChange).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove_spurious_trees1(F1a, AllTrees, I1a, I2a, Skept, Acc, I1b,
% F1b, I2b, IncChange)
% Check each tree in the forest F1a; if a tree is spurious,
% remove it and propagate the changes through I1a and I2a ^ F2a;
% if not spurious, add it to the output forest (F1b).

% 1 - terminating: all trees in forest checked.
remove_spurious_trees1([], _, I1, I2, _, IncChange, I1, [], I2,
IncChange).

% 2 - recursive: if first tree spurious, update instances and second
% forest; otherwise, just add to output forest.
remove_spurious_trees1([Tree1 | RestTrees], AllTrees, I1a, I2a,

```

```

Skept, Acc, I1c, F1, I2c, IncChange) :-
    spurious(Tree1, I1a, I2a)
    ->
    remove(Tree1, AllTrees, AllTrees2),
    update_stores(Tree1, AllTrees2, I1a, I2a, Skept, I1b, I2b,
        SubIncChange),
    NewAcc is Acc + SubIncChange,
    remove_spurious_trees1(RestTrees, AllTrees2, I1b, I2b, Skept,
        NewAcc, I1c, F1, I2c, IncChange)
    ;
    remove_spurious_trees1(RestTrees, AllTrees, I1a, I2a, Skept,
        Acc, I1c, F1T, I2c, IncChange),
    F1 = [Tree1 | F1T].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% spurious(Tree, STSInstances, LTSInstances)
% Tree is spurious with respect to the instances of a representation
% if it is groundless (i.e. contains no observed elements) and isolated
% (i.e. parent and children do not occur in any other tree).

spurious(Tree, STSInstances, LTSInstances) :-
    groundless(Tree),
    isolated(Tree, STSInstances, LTSInstances).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% groundless(Tree)
% A tree is groundless if it contains no observed elements either
% as its parent or among its children

groundless(tree(_, #_, C, _)) :-
    \+ member(@_, C).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% isolated(Tree, STSInstances, LTSInstances)
% Tree is isolated wrt Repr if the parent instance of Tree has its
% parent flag set to 'n' and all its child instances have their child
% flag set to 'n'

isolated(tree(_, P, C, _), STSI, LTSI) :-
    orphan(P, STSI, LTSI),
    childless(C, STSI, LTSI).

```



```

% Once a tree has been removed, some instances may be rendered redundant.
% These procedures update the parent and child flags of instances
% in the repr; any inferred instances which no longer have either
% parents or children are removed; observations remain in the repr
% with their new flag settings.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% update_stores(RemovedTree, AllTrees, STSI1, LTSI1, Skept, STSI2,
% LTSI2, IncChange)
% RemovedTree is a tree removed from a representation;
% OldSTS and OldLTS are updated to take account of the removal
% of the tree - parent and child flags are set, and any inferred
% instances with both flags set to 'n' are removed (not explicitly,
% but during the building of new instance lists in set_flags via
% rationalise_instances/3); IncChange tracks the incoherence change
% caused by switching of flags and removal of redundant instances;
% it also includes the IncChange caused by removing the redundant tree.

update_stores(tree(_, P, C, Inc), AllTrees, STSI1, LTSI1, Skept, STSI2,
LTSI2, ClippedIncChange) :-
    set_instances([P | C], STSI1, LTSI1, AllTrees, Skept, 0,
        STSI2, LTSI2, IncChange1),
    IncChange2 is IncChange1 - Inc,
    clip(IncChange2, ClippedIncChange).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% set_instances(RemovedTreeInstances, STSI1, LTSI1, AllTrees,
% Skept, Acc, STSI2, LTSI2, IncChange)
% For each instance in RemovedTreeInstances:
% 1. Retrieve that instance from either STSI1 or LTSI1;
% designate the remainder of the store of instances as RestStore;
% let the updated store (i.e. STSI2 or LTSI2)
% be called NewStore; let its original inc = Inc1.
% 2. Set the flags of instance wrt AllTrees.
% 3. Determine the inc of the new instance = Inc2.
% 4. Set IncChange for that instance to Inc2 - Inc1.
% 5. Recurse with the remainder of RemovedTreeInstances.

% 1 - terminating: all instances in RemovedTreeInstances have
% had their flags updated.
set_instances([], STSI, LTSI, _, _, IncChange, STSI, LTSI, IncChange).

```

```

% 2 - recursive: retrieve the next instance in
% RemovedTreeInstances from STSN and update its flags; if not retrievable,
% it must be in LTS, so retrieve it from there and update its flags;
% once retrieved, recurse with remaining instances in RemovedTreeInstances.
set_instances([I | Rest], STSI1, LTSI1, AllTrees, Skept, Acc, STSI4,
LTSI4, IncChange) :-
    strict_retrieve_instance(I, STSI1, instance(C, _, _, Inc), STSI2)
    ->
    update_instance(AllTrees, instance(C, n, n), NewRawInstance),
    rationalise_instances(NewRawInstance, STSI2, Inc, Skept,
        STSI3, SubIncChange),
    NewAcc is Acc + SubIncChange,
    set_instances(Rest, STSI3, LTSI1, AllTrees, Skept, NewAcc,
        STSI4, LTSI4, IncChange)
    ;
    strict_retrieve_instance(I, LTSI1, instance(C, _, _, Inc),
        LTSI2),
    update_instance(AllTrees, instance(C, n, n), NewRawInstance),
    rationalise_instances(NewRawInstance, LTSI2, Inc, Skept,
        LTSI3, SubIncChange),
    NewAcc is Acc + SubIncChange,
    set_instances(Rest, STSI1, LTSI3, AllTrees, Skept, NewAcc,
        STSI4, LTSI4, IncChange).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% update_instance(AllTrees, Instance, NewRawInstance)
% Updates the flags of Instance wrt Forest.

% 1 - both flags set to y, so halt
update_instance([_ | _], instance(N, Par, Chd), instance(N, y, y)) :-
    Par == y,
    Chd == y,
    !.

% 2 - no more trees to check.
update_instance([], Instance, Instance) :- !.

% 3 - next tree's parent unifies with instance, so set children flag to y.
update_instance([tree(_, I1, _, _) | RestForest], instance(I2, Par, _),
UpdatedInstance) :-

```

```

    I1 == I2,
    update_instance(RestForest, instance(I2, Par, y), UpdatedInstance), !.

% 4 - next tree contains instance as a child, so set parent flag to y.
update_instance([tree(ID, _, C, _) | RestForest], instance(N, _, Chd),
UpdatedInstance) :-
    indexed(ID, #_, N),
    strict_child_present(N, C),
    update_instance(RestForest, instance(N, y, Chd),
        UpdatedInstance), !.

% 5 - none of the above apply, so ignore the next tree
update_instance([_ | RestForest], Instance, UpdatedInstance) :-
    update_instance(RestForest, Instance, UpdatedInstance).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% REMOVING UNNECESSARY INSTANCES %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rationalise_instances(Instance, CurrentInstances, Inc, Skept,
% NewInstances, IncChange)
% If Instance is inferred and both flags set to 'n', it isn't added to
% NewInstances; otherwise, it is, providing it doesn't occur elsewhere
% in CurrentInstances; Inc is the original Inc of the instance;
% if it is removed, IncChange is - Inc; otherwise, IncChange is
% NewInc - Inc. Better to determine the inc of instances here,
% as it saves finding inc for instances which are going to be
% removed anyway.

% 1 - terminating: if instance is inferred and occurs in no trees, remove it.
rationalise_instances(instance(#_, n, n), Instances, Inc, _,
Instances, ClippedIncChange) :-
    IncChange is 0 - Inc,
    clip(IncChange, ClippedIncChange), !.

% 2 - terminating: instance has child or parent flag set to y,
% so add to output if it is not duplicated in the list of
% CurrentInstances; otherwise, return original list and incchange = 0
rationalise_instances(instance(C, Par, Chd), Instances, Inc, Skept,
NewInstances, ClippedIncChange) :-

```



```

% transfer_tree(repr(STSExgraph, LTSExgraph, Inc), Skept,
% repr(NewSTSExgraph, NewLTSExgraph, NewInc))
% Transfers elts from STS to LTS, reducing incoherence of STS
% in the process.
% Transfer is initially tree-based, and follows these steps:
% 1. Select the tree in STS with lowest incoherence;
% 2. Find all instances dependent on that tree;
% 3. Transfer the tree and its dependent instances to LTS.
% An instance is dependent on a tree T in STS if it is a child of
% T and the child of no other tree in STS.

transfer_tree(repr(STSI1 ^ STSF1, LTSI1 ^ LTSF1, _), Skept,
repr(NewSTS, LTSI2 ^ LTSF2, NewInc)) :-
    select_transfer_tree(STSF1, Tree, STSF2),
    dependent_instances(Tree, STSF2, STSI1, STSI2, Dependents),
    move_to_LTS(Dependents, LTSI1, LTSI2),
    move_to_LTS([Tree], LTSF1, LTSF2),
    inc(STSI2 ^ STSF2, Skept, NewSTS, NewInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% select_transfer_tree(Forest1, Tree, Forest2)
% Selects the tree Tree from Forest1 with lowest inc, with Forest2
% as the remainder.
% Sets the first tree in Forest1 as the 'sofar', then attempts to
% find a tree with lower inc using an accumulator.
% N.B. have to select it then remove it to prevent forest ordering
% being disrupted

select_transfer_tree([Tree | Rest], Selected, Forest2) :-
    select_transfer_tree1(Rest, Tree, Selected),
    remove(Selected, [Tree | Rest], Forest2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% select_transfer_tree1(Forest, SelectedTreeSofar, SelectedTree)
% Selects the least incoherent tree as the tree to be transferred;
% if two trees have equal inc, the oldest one (i.e. created earliest) is
% selected as the transfer tree.

% 1 - terminating: Forest is empty, so best tree found Sofar is
% SelectedTree
select_transfer_tree1([], Sofar, Sofar).

```

```

% 2 - recursive: if next tree has equal to or lower inc than the
% current Sofar, make it the sofar; otherwise, retain the current sofar.
% This should ensure that if trees are equally incoherent, oldest is
% transferred first.
select_transfer_tree1([tree(ID1, P1, C1, Inc1) | Rest],
tree(ID2, P2, C2, Inc2), Selected) :-
    Inc1 =< Inc2
->
    select_transfer_tree1(Rest, tree(ID1, P1, C1, Inc1), Selected)
;
    select_transfer_tree1(Rest, tree(ID2, P2, C2, Inc2), Selected).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% dependent_instances(Tree, Forest, Instances, RestInstances,
% DependentInstances)
% Tree = tree selected for transfer
% Forest = trees currently in STS (other than Tree)
% Instances = list of instances in STS
% DependentInstances = instances in Instances which occur in a
% parented tree and not anywhere else in Forest
% RestInstances = Instances with all DependentInstances removed
% (i.e. instances which will remain in STS after transfer).

dependent_instances(tree(_, Parent, Children, _), Forest, Instances1,
Instances2, Dependents) :-
    dependent_on_tree([Parent | Children], Instances1, Forest, [],
        Instances2, Dependents).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% dependent_on_tree(TreeInstances, Instances1, Forest, Sofar,
% Instances2, TreeDependents)
% TreeInstances = the parent and children of a tree to be transferred
% to LTS. Each element of TreeInstances is checked against Forest -
% If the element occurs in no other tree and the corresponding instance
% is in STS, it is removed from Instances1 and added to the sofar list;
% If the element occurs in no other tree and is not in STS, it is ignored;
% If it occurs in another tree, it is not dependent on the tree,
% and remains in Instances1.

% 1 - terminating: all elements of tree have been checked.

```

```

dependent_on_tree([], Instances, _, Dependents, Instances, Dependents).

% 2 - recursive: if the next element's corresponding instance is in
% STS, then check whether it can be transferred (i.e. whether it
% occurs in another tree in STS); if the corresponding instance is
% not in STS, then that element is ignored, as it cannot be transferred
% (it's already in LTS).
dependent_on_tree([Elt | RestElts], Instances1, Forest, Sofar,
InstancesLeft, TreeDependents) :-
    strict_retrieve_instance(Elt, Instances1, Retrieved, Instances2)
    ->
    transferrable(Forest, Retrieved, Instances2, Instances3,
        Sofar, NewSofar),
    dependent_on_tree(RestElts, Instances3, Forest, NewSofar,
        InstancesLeft, TreeDependents)
    ;
    dependent_on_tree(RestElts, Instances1, Forest, Sofar,
        InstancesLeft, TreeDependents).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% transferrable(Forest, Instance, CurrentInstances, NewInstances,
% DependentsSofar, NewDependentsSofar)
% If RetrievedInstance occurs in Forest, it is not transferrable, and the
% NewDependentsSofar = DependentsSofar
% and NewInstances = [RetrievedInstance | CurrentInstances];
% Else, RetrievedInstance is transferrable, and is added to the
% sofar list and NewInstances = CurrentInstances.
% So, once the instance has been retrieved, this procedure
% decides whether it can be added to the list of instances to be
% transferred, or whether it has to remain in STS.

% 1 - terminating: RetrievedInstance hasn't strictly matched a
% child of any tree in Forest, so can be added to the list of dependents.
transferrable([], instance(Content, Par, Chd, Inc), Instances,
Instances, Sofar, [instance(Content, Par, Chd, Inc) | Sofar]).

% 2 - terminating: RetrievedInstance strictly matches a child of
% the next tree in Forest, so isn't transferrable.
transferrable([tree(_, _, Children, _) | _], instance(Content, y, Chd, Inc),
Instances, [instance(Content, y, Chd, Inc) | Instances], Sofar, Sofar) :-
    strict_child_present(Content, Children), !.

```



```

% otherwise, calculates inc and annotates the instance.

instances_inc(InstancesIn, Skept, InstancesOut, InstancesInc) :-
    instances_inc(InstancesIn, Skept, InstancesOut, 0, InstancesInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% instances_inc(InstancesIn, Skept, InstancesOut, Acc, InstancesInc)

% 1 - terminating: no more trees to process
instances_inc([], _, [], InstancesInc, InstancesInc).

% 2 - recursive: next instance has already been assigned incoherence
instances_inc([instance(Content, Par, Chd, Inc) | RestInstances], Skept,
[instance(Content, Par, Chd, Inc) | RestOut], Acc, InstancesInc) :-
    NewAcc is Acc + Inc,
    instances_inc(RestInstances, Skept, RestOut, NewAcc, InstancesInc).

% 3 - recursive for 'raw' instances (i.e. those without incoherence calculated)
instances_inc([instance(Content, Par, Chd) | RestInstances], Skept,
[NewInstance | RestOut], Acc, InstancesInc) :-
    instance_inc(instance(Content, Par, Chd), Skept, NewInstance, Inc),
    NewAcc is Acc + Inc,
    instances_inc(RestInstances, Skept, RestOut, NewAcc, InstancesInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% trees_inc(TreesIn, Skept, TreesOut, TreesInc)
% Determines the inc of a forest; each tree will already have been assigned
% its individual incoherence when it was created, so this just adds these
% values together. Alternatively, if a tree has no incoherence assigned,
% this procedure can assign it; the trees are output in annotated form.

trees_inc(TreesIn, Skept, TreesOut, TreesInc) :-
    trees_inc(TreesIn, Skept, TreesOut, 0, TreesInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% trees_inc(Trees, Skept, TreesOut, Acc, TreesInc)

% 1 - terminating: no more trees to process
trees_inc([], _, [], TreesInc, TreesInc).

% 2 - recursive: next tree has already been assigned incoherence

```

```

trees_inc([tree(ID, P, C, Inc) | RestTrees], Skept,
[tree(ID, P, C, Inc) | RestOut], Acc, TreesInc) :-
    NewAcc is Acc + Inc,
    trees_inc(RestTrees, Skept, RestOut, NewAcc, TreesInc).

% 3 - recursive for 'raw' trees (i.e. those without incoherence calculated)
trees_inc([tree(ID, P, C) | RestTrees], Skept, [NewTree | RestOut],
Acc, TreesInc) :-
    tree_inc(tree(ID, P, C), Skept, NewTree, Inc),
    NewAcc is Acc + Inc,
    trees_inc(RestTrees, Skept, RestOut, NewAcc, TreesInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ONE INSTANCE INC PROCEDURES %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Used to determine inc caused by those instances which are at the
% roots or leaves of the exgraph; NB has nothing to do with alternative
% explanations, alternative elaborations, and so on - these are dealt
% with by processing the trees.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% instance_inc(Instance, Skept, AnnotatedInstance, InstanceInc)

instance_inc(instance(Instance, Parents, Children), Skept,
instance(Instance, Parents, Children, InstanceInc2), InstanceInc2) :-
    lookup_node(Instance, Inf, Ubi),
    value(Instance, Parents, Ubi, Skept, Xinc),
    value(Instance, Children, Inf, Skept, Einc),
    InstanceInc is Xinc + Einc,
    clip(InstanceInc, InstanceInc2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% value(Instance, Present, Value, Skept, InstanceInc)
% Used to determine both types of inc (xinc and einc) for individual
% instances.
% Instance = @_ (observed) or #_ (inferred)
% Present is either a y or n, denoting whether an instance has
% parents or children, depending on context;
% Value is either inf or ubi, depending on context.

```

```

% 1 - observed instance with parents or children not present
value(@_, n, Value, _Skept, InstanceInc) :-
    InstanceInc is Value - 1.

% 2 - observed instance with parents or children present
value(@_, y, _, _, 0).

% 3 - inferred instance with parents or children not present
value(#_, n, Value, Skept, InstanceInc) :-
    InstanceInc is Value * Skept.

% 4 - inferred instance with parents or children present
value(#_, y, _, _, 0).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ONE TREE INC PROCEDURES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% tree_inc(Tree, Skept, AnnotatedTree, TreeInc)
% Determines the incoherence caused by missing trees, based on
% which tree is present.
% Used to determine the incoherence of a tree of form
%     tree(ID, Parent, Children, CurrentInc)
% Returns the tree unchanged except for its inc.

tree_inc(tree(ID, P, C), Skept, tree(ID, P, C, TreeInc2), TreeInc2) :-
    alt_expls_inc(tree(ID, P, C), Skept, AltExplsInc),
    alt_elabs_inc(tree(ID, P, C), Skept, AltElabsInc),
    TreeInc is AltExplsInc + AltElabsInc,
    clip(TreeInc, TreeInc2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ALTERNATIVE EXPLANATIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% alt_expls_inc(Tree, Skept, AltExplsInc)
% Finds all alternative explanations of the observed children of Tree

% 1 - if no inferred children, schema is completed, so other possible expls

```

```

% have even less influence
alt_expls_inc(tree(ID, Parent, Children), Skept, AltExplsInc2) :-
    member(#_, Children)
    ->
    other_parents(Children, ID, OtherExpls),
    other_parents_inc(OtherExpls, ID, Parent, Skept, AltExplsInc2)
    ;
    other_parents(Children, ID, OtherExpls),
    other_parents_inc(OtherExpls, ID, Parent, Skept, AltExplsInc1),
    AltExplsInc2 is AltExplsInc1 * Skept, !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% other_parents(Children, ID, OtherPossParents)
% OtherPossParents is the set of schema parents which could
% potentially explain the observed Children of a tree.
% Sets up the accumulator with all possible explanations of the
% first child and calls other_parents1/4.

% 1 - no children are observed, so OtherPossParents is empty
other_parents([], _, []).

% 2 - if the first child is observed, set up accumulator.
other_parents([@H | T], ID1, OtherExpls) :-
    find_one_parents(H, ID1, OneParents),
    other_parents1(T, OneParents, ID1, OtherExpls).

% 3 - if first child is inferred, ignore it.
other_parents([#_ | T], ID1, OtherExpls) :-
    other_parents(T, ID1, OtherExpls).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% other_parents1(Children, Acc, ID, OtherPossParents)
% The accumulator initially contains the other possible parents
% of the first child, connected to the IDs of the schemas which
% could have been used.

% 1 - terminating: unify the list of other possible parents with
% the sofar list.
other_parents1([], PossParents, _, PossParents).

% 2 - recursive: find all possible parents of the next observed

```

```

% child, and intersect this with the list of possible parents
% of previous children.
other_parents1([@H | T], Acc, ID1, OtherPossParents) :-
    find_one_parents(H, ID1, OneParents),
    intersection(OneParents, Acc, NewAcc),
    other_parents1(T, NewAcc, ID1, OtherPossParents).

% 3 - recursive: next child inferred, so ignore it.
other_parents1([#_ | T], Acc, ID, OtherPossParents) :-
    other_parents1(T, Acc, ID, OtherPossParents).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% find_one_parents(Child, ParentID, ChildsOtherParents)
% Finds the other possible parents of child, given the actual ParentID

find_one_parents(H, ID1, OneParents) :-
    findall([ID2, P],
            (index(ID2, P, H), ID2 \== ID1),
            OneParents).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% intersection(L1, L2, L3)
% L3 is the intersection between L1 and L2;
% Note that this also strips off the IDs of possible parents, leaving
% just the node.

% 1 - terminating: no more elts in first list
intersection([], _, []).

% 2 - terminating: no more elts in second list
intersection(_, [], []).

% 3 - next element of the first list occurs in the second list, so can be
% added to the list of intersections
intersection([[ID, P] | T1], [[ID, P] | T2], [[ID, P] | T3]) :-
    intersection(T1, T2, T3), !.

% 4 - next element of first list doesn't match the next element of
% second list, and has a smaller id, so add it as a possible parent
intersection([[ID1, _] | T1], [[ID2, P2] | T2], L3) :-
    ID1 < ID2,

```

```

        intersection(T1, [[ID2, P2] | T2], L3).

% 5 - next element of second list doesn't match the next element
% of first list, and has a smaller id
intersection([[ID1, P1] | T1], [[ID2, _] | T2], L3) :-
    ID1 > ID2,
    intersection([[ID1, P1] | T1], T2, L3).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% other_parents_inc(OtherExpls, ID, Parent, Skept, AltExplsInc)
% Sets up accumulator

other_parents_inc(OtherExpls, ID, Parent, Skept, AltExplsInc) :-
    other_parents_inc(OtherExpls, ID, Parent, Skept, 0, AltExplsInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% other_parents_inc(OtherExpls, ID, Parent, Skept, Acc, AltExplsInc)
% For each member E of OtherExpls:
% If E is indexed to Parent
%     inc(E) = 0
% Else
%     Lookup ubi of E
%     If Parent observed
%         inc(E) = ubi(E) * Skept/2
%     Else (Parent inferred)
%         inc(E) = ubi(E) * (Skept + 0.5)
%     Endif
% Endif
%
% OtherExpls = list of [PossID, Parent] pairs
% ID = ID of actual explanation

% 1 - terminating: no more other expls to add
other_parents_inc([], _, _, _, AltExplsInc, AltExplsInc).

% 2 - if next member of OtherExpls is indexed to Parent, inc = 0;
% otherwise, lookup the ubi of E wrt parent's type token (# or @)
other_parents_inc([[_, OtherExpl] | T], ID, Parent, Skept, Acc,
AltExplsInc) :-
    indexed(ID, Parent, #OtherExpl)
    ->

```



```

    other_parents_inc(T, ID, Parent, Skept, Acc, AltExplsInc)
    ;
    other_expl_inc(Parent, OtherExpl, Skept, OneInc),
    NewAcc is Acc + OneInc,
    other_parents_inc(T, ID, Parent, Skept, NewAcc, AltExplsInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% other_expl_inc(Parent, OtherExpl, Skept, OneInc)

% 1 - if actual Parent is observed
other_expl_inc(@_, OtherExpl, Skept, OneInc) :-
    node(OtherExpl, _, Ubi),
    OneInc is Ubi * Skept / 2.

% 2 - if actual Parent is inferred
other_expl_inc(#_, OtherExpl, Skept, OneInc) :-
    node(OtherExpl, _, Ubi),
    OneInc is Ubi * (Skept + 0.5).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ALTERNATIVE ELABORATION SETS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% alt_elabs_inc(Tree, Skept, AltElabsInc)
% Determines the Inc caused by alternative elaborations:
% (NB this gives the informativity caused by possible elaborations that
% aren't under the schema ID used to create the exgraph)

alt_elabs_inc(tree(ID, Parent, Children), Skept, AltElabsInc) :-
    poss_elabs(ID, Parent, PossElabs),
    sum_informativity(PossElabs, PossElabsInf),
    lookup_node(Parent, Inf, _),
    BasicInc is Inf - PossElabsInf,
    alt_elabs_inc1(Children, BasicInc, Skept, AltElabsInc).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% poss_elabs(ID, Parent, PossElabs)
% PossElabs is the set of possible elaborations of Parent, wrt schema
% identified by ID

```

```

poss_elabs(ID, Parent, PossElabs) :-
    raw_content(Parent, P),
    ID: P / _ ---> PossElabs.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% alt_elabs_inc1(Children, BasicInc, Skept, AltElabsInc)
% Applies Skept to BasicInc, according to whether any member of Children
% has been observed (this gives lower AltElabsInc)
% The figure by which Skept is multiplied determines (generally) how
% eager the system is to make elaborative inferences: the closer it is
% to 1, the more reticent it is; when value = 1, elaborative inferences
% which aren't based on the presence of existing observations cease
% to be made, as the elaboration has the same inc as a childless
% inferred instance. Raising this value reduces the amount of top-down
% inference from inferred instances.

alt_elabs_inc1(Children, BasicInc, Skept, AltElabsInc) :-
    member(@_, Children)
    ->
        AltElabsInc is 0
    ;
        AltElabsInc is BasicInc * Skept * 0.8.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sum_informativity(NodeList, SumInf)
% calls accumulator

sum_informativity(NodeList, SumInf) :-
    sum_informativity(NodeList, 0, SumInf).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% sum_informativity(NodeList, Acc, SumInf)
% NB these are raw nodes, not marked with # or @

% 1 - terminating: all nodes processed
sum_informativity([], SumInf, SumInf).

% 2 - recursive
sum_informativity([H/_ | T], Acc, SumInf) :-
    node(H, Inf, _),
    NewAcc is Acc + Inf,

```

```
sum_informativity(T, NewAcc, SumInf).
```

C.4 *shared.pl*

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ADMINISTRATION AND SHARED PROCEDURES %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
CHECKING CONSTRAINTS %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% check(Constraints)
% Calls each constraint in turn to pass variable bindings along the rule.
% This ensures that relations are dependent on the events they connect, and
% prevents inferring event tokens from the presence of unrelated relations.

check([]).

check([H | T]) :-
    call(H),
    check(T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% X1 ==> X2
% Binds variables between two terms, from the term on the left of the
% arrow to the one on the right (directed unification).

% 1 - two variables
X1 ==> X2 :-
    var(X1),
    var(X2),
    X1 = X2, !.

% 2 - atom and variable
X1 ==> X2 :-
    \+ X1 == [],
    atomic(X1),
    var(X2),
    X1 = X2, !.

% 3 - two non-variables
X1 ==> X2 :-

```

```

X1 == X2.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% combine_raw_content(InstanceRawContent1, InstanceRawContent2)
% Combines the raw content of two nodes, ensuring that bindings go
% only from InstanceContent1 to InstanceContent2.
% (Raw content means that the status symbols have
% been stripped from the content, i.e. the '@' and '#'.)
% InstanceContent2 contains the output, once unification has been checked.

combine_raw_content(event(T1, Pred, Args), event(T2, Pred, Args)) :-
    T1 ==> T2.

combine_raw_content(goal(T1, Pred, Args), goal(T2, Pred, Args)) :-
    T1 ==> T2.

combine_raw_content(relation(Pred, Args1), relation(Pred, Args2)) :-
    Args1 == Args2.

combine_raw_content(habit(Pred, Args), habit(Pred, Args)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% STRUCTURE ACCESS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% NB loose procedures combine the content of the instance to be
% retrieved with the content used for retrieval; strict procedures
% only match the content of two instances

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% loose_remove_duplicates(Items1, Items2)
% Items2 is Items1 with all duplicated items removed;

% 1 - terminating
loose_remove_duplicates([], []).

% 2 - recursive: FOR INSTANCES
% if next item is duplicated in the tail of the input instance list,
% remove it from consideration; else, add next item to the output
% and recurse with the tail of the first list
loose_remove_duplicates([instance(C, Par, Chd, Inc) | T], List2) :-

```



```

% tree_duplicated(Tree, Forest1, Forest2)
% Forest2 is Forest1 minus the duplication of Tree

% 1 - terminating
tree_duplicated(tree(ID, Parent1, _, Inc),
[tree(ID, Parent2, _, Inc) | Rem], Rem) :-
    copy_term(Parent1, Parent1Copy),
    copy_term(Parent2, Parent2Copy),
    loose_match(Parent1Copy, Parent2Copy), !.

% 2 - recursive: try a later item in List1
tree_duplicated(Item, [H | T], [H | Rem]) :-
    tree_duplicated(Item, T, Rem).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% strict_retrieve_instance(Content, Instances1, RetrievedInstance,
% Instances2)
% Instances2 is Instances1 minus RetrievedInstance; the content of
% RetrievedInstance exactly matches Content. Doesn't update
% status or variable bindings.

% 1 - terminating: next instance's content exactly matches Content
strict_retrieve_instance(Content1, [instance(Content2, Par, Chd, Inc) | Rest],
instance(Content2, Par, Chd, Inc), Rest) :-
    Content1 == Content2, !.

% 2 - recursive: try later instances for exact match.
strict_retrieve_instance(Content1, [Instance | Rest1], Retrieved,
[Instance | Rest2]) :-
    strict_retrieve_instance(Content1, Rest1, Retrieved, Rest2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% loose_retrieve_instance(ContentToRetrieve, Instances1,
% RetrievedInstance, Instances2),
% Instances2 is Instances1 minus RetrievedInstance; the content of
% RetrievedInstance can be combined with the content of ContentToRetrieve.
% Also note that this updates the status of the retrieved instance,
% e.g. from '#' to '@'.

% 1 - terminating: next instance in instance list is observed,
% and can be combined with Content1.

```

```

loose_retrieve_instance(Content1,
[instance(@Content2, Par, Chd, Inc) | RestInstances],
instance(@Content1, Par, Chd, Inc), RestInstances) :-
    combine_content(Content2, Content1), !.

% 2 - terminating: next instance in instance list is inferred,
% and can be combined with Content1.
loose_retrieve_instance(Content1,
[instance(#Content2, Par, Chd, Inc) | RestInstances],
instance(#Content1, Par, Chd, Inc), RestInstances) :-
    combine_content(Content2, Content1), !.

% 3 - recursive: next instance doesn't combine, so recurse with rest
% of instance list.
loose_retrieve_instance(Content, [Instance | RestInstances1],
RetrievedInstance, [Instance | RestInstances2]) :-
    loose_retrieve_instance(Content, RestInstances1,
        RetrievedInstance, RestInstances2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% tree_present(Tree, Forest)

% 1 - terminating: Tree has same ID and matching parent as next tree in
% Forest.
tree_present(tree(ID, P1, _, _), [tree(ID, P2, _, _) | _]) :-
    loose_match(P1, P2), !.

% 2 - terminating: as 1, but for trees not assigned incoherence yet
tree_present(tree(ID, P1, _, _), [tree(ID, P2, _) | _]) :-
    loose_match(P1, P2), !.

% 3 - recursive
tree_present(Tree, [_ | T]) :-
    tree_present(Tree, T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% strict_child_present(Child, Children)
% Child must exactly match a member of Children, i.e. have the same
% token and variable bindings

% 1 - terminating: next element of Children matches Child.

```



```

strict_child_present(Child1, [Child2 | _]) :-
    Child1 == Child2, !.

% 2 - recursive
strict_child_present(Child, [_ | T]) :-
    strict_child_present(Child, T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% loose_child_present(Child, Children)
% Child is instance content annotated with @ or #; children is a
% list of the same format elements.

% 1 - terminating: next element of Children matches Child.
loose_child_present(Child1, [Child2 | _]) :-
    loose_match(Child1, Child2), !.

% 2 - recursive
loose_child_present(Child, [_ | T]) :-
    loose_child_present(Child, T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% loose_match(Instance1, Instance2)
% Matches two instances, which may or may not be inferred.
% NB the content of the instance must be the same, regardless of
% whether both or either are inferred.

% 1 - covers cases where both are observed, or both inferred
loose_match(I, I).

% 2 - first instance inferred
loose_match(#I, @I).

% 3 - second instance inferred
loose_match(@I, #I).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% select_instance(Which?, Instances, SelectedInstance, OtherInstances)
% Which? = 'least_inc' for the instance in Instances with lowest inc,
% or 'most_inc' for the instance with the highest inc, or 'any'.
% Calls accumulator with first instance as sofar.

```

```

% 1a - any instance, so select the first one
select_instance(any, [Instance | Rest], Instance, Rest).

% 1b - any instance from further down the list
select_instance(any, [Instance | Rest1], Selected, [Instance | Rest2]) :-
    select_instance(any, Rest1, Selected, Rest2).

% 2 - first instance in instance list
select_instance(first, [Instance | Rest], Instance, Rest).

% 3 - most or least inc: call sub-procedure
select_instance(Which, [Instance | Rest], Selected, OtherInstances) :-
    (Which == most_inc; Which == least_inc),
    select_instance(Rest, Which, Instance, Selected, OtherInstances).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% select_instance(Instances, Sofar, SelectedInstance, OtherInstances)

% 1 - terminating: no more instances, so set sofar instance to
% SelectedInstance
select_instance([], _, Selected, Selected, []).

% 2 - recursive: next instance has higher incoherence, so make
% it sofar and put current sofar onto output.
select_instance([NextInstance | Rest], Which, Sofar, Selected,
[Output | Other]) :-
    compare_incs(Which, NextInstance, Sofar, Output, NewSofar),
    select_instance(Rest, Which, NewSofar, Selected, Other).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% compare_incs(Which, Instance1, Sofar, Output, NewSofar)
% Compares the inc of Instance1 and Sofar; depending on the instantiation
% of Which, will put the instance with lowest or highest inc into
% NewSofar, and the other instance into Output

compare_incs(least_inc, instance(C1, Par1, Chd1, Inc1),
instance(C2, Par2, Chd2, Inc2), instance(C1, Par1, Chd1, Inc1),
instance(C2, Par2, Chd2, Inc2)) :-
    Inc2 =< Inc1, !.

compare_incs(least_inc, instance(C1, Par1, Chd1, Inc1),

```

```
instance(C2, Par2, Chd2, Inc2), instance(C2, Par2, Chd2, Inc2),
instance(C1, Par1, Chd1, Inc1)).
```

```
compare_incs(most_inc, instance(C1, Par1, Chd1, Inc1),
instance(C2, Par2, Chd2, Inc2), instance(C1, Par1, Chd1, Inc1),
instance(C2, Par2, Chd2, Inc2)) :-
    Inc2 >= Inc1, !.
```

```
compare_incs(most_inc, instance(C1, Par1, Chd1, Inc1),
instance(C2, Par2, Chd2, Inc2), instance(C2, Par2, Chd2, Inc2),
instance(C1, Par1, Chd1, Inc1)).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% raw_content(Content, RawContent)
```

```
% Returns the raw content of an instance's content, i.e. strips off
% the # or @ symbol.
```

```
raw_content(#I, I).
```

```
raw_content(@I, I).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% indexed(ID, I1, I2)
```

```
% Determines whether there is an index that links instances I1 and I2,
% regardless of whether they are inferred or observed.
```

```
% 1
```

```
% Both instances inferred
```

```
indexed(ID, #I1, #I2) :-
    index(ID, I1, I2).
```

```
% 2
```

```
% First instance inferred
```

```
indexed(ID, #I1, @I2) :-
    index(ID, I1, I2).
```

```
% 3
```

```
% Second instance inferred
```

```
indexed(ID, @I1, #I2) :-
    index(ID, I1, I2).
```



```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% OUTPUT PROCEDURES %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% WRITING INDICES and NODES TO SCREEN %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% show nodes and indexes
show :-
    shown,
    showi.

% show nodes
shown :-
    findall(node(Node, Inf, Ubi), node(Node, Inf, Ubi), AllNodes),
    wl(AllNodes).

% show indexes
showi :-
    findall(index(ID, LH, RH), index(ID, LH, RH), AllIndices),
    wl(AllIndices).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% WRITING TO STREAM %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_status(Cycle, Skept, Tolerance, Range)
% Writes the current status of comprehension process to screen.

write_status(Cycle, Skept, Tolerance, Range) :-
    write('Cycle number = '), write(Cycle), nl,
    write('Skepticism = '), write(Skept), tab(10),
    write('Tolerance = '), write(Tolerance), nl,
    write('Range = '), write(Range), nl, nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_exts(Extensions)
% An extension has the form

```

```

%      ext(Repr, Rating)
% where Rating is incoherence change.

write_exts([ext(Repr, IncChange)]) :-
    write_extension(ext(Repr, IncChange)),
    write('ALL EXTENSIONS WRITTEN'), nl,
    write_line, nl, nl,
    continue(Repr).

write_exts([ext(Repr, IncChange) | Rest]) :-
    write_extension(ext(Repr, IncChange)),
    write('PRESS RETURN FOR NEXT EXTENSION'), nl,
    continue(Repr),
    write_exts(Rest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_extensions_to_file(Extensions)
% Same as write_extensions, but without prompts.

write_extensions_to_file([]).

write_extensions_to_file([ext(Repr, IncChange) | Rest]) :-
    write_extension(ext(Repr, IncChange)),
    write_extensions_to_file(Rest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_extension(ext(Repr, Rating))
% Write a single extension to output stream.

write_extension(ext(Repr, Rating)) :-
    write_repr(Repr),
    write_rating(Rating), nl,
    nl,
    write('*****'),
    nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_rating(Rating)
% Rating is a number (i.e. reduction in incoherence) or a message
% describing how an observation has been added to the representation
% or which r-elt was transferred

```

```

% 1 - if rating describes observation
write_rating(obs(Message, _IncChange)) :-
    write('Observation incorporated by '), write(Message).

% 2 - if rating describes transfer
write_rating(transfer(Rating)) :-
    write('Transfer changed incoherence by '), write(Rating).

% 3 - if rating describes inference
write_rating(infer(Method, Created, Removed)) :-
    TotalIncChange is Created + Removed,
    write('Inferred new elements'), nl,
    write_method(Method), nl,
    write('Total inc change = '), write(TotalIncChange), nl,
    write('Incchange due to creation = '), write(Created), nl,
    write('Incchange due to removal = '), write(Removed).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_method(Method)
% Method is either top_down or bottom_up, depending on the inference
% method used

write_method(top_down) :-
    write('Elements created top-down').

write_method(bottom_up) :-
    write('Elements created bottom-up').

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_repr(Repr)
% Writes a single representation to screen.

write_repr(repr(STSExgraph, LTSExgraph, Inc)) :-
    nl,
    write('----- STS -----'),
    write_exgraph(STSExgraph),
    nl,
    write('----- LTS -----'),
    write_exgraph(LTSExgraph),
    nl, nl,

```

```

        write('----- Incoherence = '), write(Inc), write(' -----'),
        nl, nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_exgraph(Instances ^ Forest)
% Writes a single exgraph to screen.

write_exgraph(Instances ^ Forest) :-
    nl,
    write('Instances: '), write(Instances), nl, nl,
    write('Forest: '), nl, nl,
    write_forest(Forest).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_forest(Forest)
% Writes a list of trees to screen. If the list of trees is empty, writes
% NO TREES to screen.

write_forest([]) :-
    write('NO TREES').

write_forest([H | T]) :-
    write_forest1([H | T]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_forest1(Forest)
% Called by write_forest/1.

write_forest1([]).

write_forest1([H | T]) :-
    write(H), nl, nl,
    write_forest1(T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_line
% writes a line of asterisks to screen, to separate processing cycles

write_line :-
    write('*****'),
    nl.

```



```

%%%%%%%%%% PREDICATES FOR WRITING NODES AND INDICES TO FILE %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% write_clauses_to_stream(TermList)
% writes a list of Prolog clauses to the current output stream

% 1 - terminating: all terms in the list have been written to the
% output stream
write_clauses_to_stream([]).

% 2 - recursive: write the first term in the list to the output stream
% then call write_clauses_to_file recursively on the rest of the terms
% in the list
write_clauses_to_stream([H | T]) :-
    portray_clause(H),
    write_clauses_to_stream(T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% LIST PROCESSING %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% append(L1,L2,L3)
append([], L, L).
append([H | T], L2, [H | NewT]) :-
    append(T, L2, NewT).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% member(Elt, List) : true if Elt occurs as an element of List
member(Elt, [Elt|_]) :- !.
member(Elt, [_|T]) :- member(Elt, T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% strict_member(Elt, List) : true if Elt matches an element of List
strict_member(Elt1, [Elt2|_]) :-
    Elt1 == Elt2, !.
strict_member(Elt, [_|T]) :- strict_member(Elt, T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove_duplicates(Untidy_List, Tidy_List)

```

```

% takes all duplicates out of a list
% e.g. remove_duplicates([a,a,b,b,c], Tidy)
% Tidy = [a,b,c]

remove_duplicates(List, Tidy_List) :-
    remove_duplicates(List, [], Tidy_List).

% remove_duplicates/3
% second argument is an accumulator

remove_duplicates([], Sofar, Sofar).

remove_duplicates([H | T], Sofar, Tidy) :-
    member(H, T)
    ->
    remove_duplicates(T, Sofar, Tidy)
    ;
    remove_duplicates(T, [H | Sofar], Tidy).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove(Elt, List, NewList)
% NewList is List with Elt removed

remove(Elt, [Elt | Tail], Tail).

remove(Elt, [H | Tail], [H | NewList]) :-
    \+ Elt = H,
    remove(Elt, Tail, NewList).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% remove_sublist(Sublist, List, Remainder)

% 1 - terminating: no more elements in Sublist
remove_sublist([Item], List, NewList) :-
    remove(Item, List, NewList).

% 2 - recursive: remove first element of Sublist from List, then recurse
% with tail of Sublist
remove_sublist([Head | Tail], List, Remainder) :-
    remove(Head, List, NewList),
    remove_sublist(Tail, NewList, Remainder).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% wl(List)
% Writes each element of List onto a separate line

wl([]).

wl([H | T]) :-
    write(H),
    nl,
    wl(T).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% same_members(L1, L2)
% L1 has the same members as L2 (not necessarily in the same order).

same_members([], []).

same_members([H1 | T1], L2) :-
    remove(H1, L2, L3),
    same_members(T1, L3).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% strict_remove(Elt, List, NewList)
% NewList is List with Elt removed; Elt must strictly match a member
% of List

strict_remove(Elt1, [Elt2 | Tail], Tail) :-
    Elt1 == Elt2.

strict_remove(Elt, [H | Tail], [H | NewList]) :-
    Elt \== H,
    strict_remove(Elt, Tail, NewList).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% max_list(NumList, Max)
% Max is the largest number in the list of numbers NumList

max_list([], 0).

max_list([H | T], Max) :-

```

```
max_list(T, H, Max).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% max_list(NumList, Acc, Max)
% used by max_list/2

max_list([], Max, Max).

max_list([H | T], Acc, Max) :-
    Acc > H
    ->
    max_list(T, Acc, Max)
    ;
    max_list(T, H, Max).
```

Bibliography

- [Aha et al., 1991] Aha, D., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1):37–66.
- [Allen, 1984] Allen, J. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154.
- [Alterman, 1985] Alterman, R. (1985). A dictionary based on concept coherence. *Artificial Intelligence*, 25(2):153–186.
- [Alterman, 1991] Alterman, R. (1991). Understanding and summarization. *Artificial Intelligence Review*, 5(4):239–254.
- [Alterman and Bookman, 1990] Alterman, R. and Bookman, L. (1990). Some computational experiments in summarisation. *Discourse Processes*, 13(2):143–174.
- [Alterman and Bookman, 1992] Alterman, R. and Bookman, L. (1992). Reasoning about a semantic memory encoding of the connectivity of events. *Cognitive Science*, 16(2):205–232.
- [Anderson, 1983] Anderson, J. (1983). *The Architecture of Cognition*. Harvard University Press, Cambridge, MA.
- [Appelt, 1990] Appelt, D. (1990). A theory of abduction based on model preferences. In *Working Notes of the 1990 Spring Symposium on Automated Abduction*, pages 67–71, University of California, Irvine.
- [Asher, 1993] Asher, N. (1993). *Reference to Abstract Objects in Discourse*. Kluwer Academic Publishers, Dordrecht, Netherlands.
- [Baddeley, 1992] Baddeley, A. (1992). Is working memory working?: The 15th Bartlett lecture. *Quarterly Journal of Experimental Psychology*, 44A(1):1–31.

- [Bar-Hillel and Carnap, 1964] Bar-Hillel, Y. and Carnap, R. (1964). An outline of a theory of semantic information. In Bar-Hillel, Y., editor, *Language and Information: Selected Essays on their Theory and Application*, pages 221–274. Jerusalem Academic Press, Jerusalem.
- [Barsalou, 1992] Barsalou, L. (1992). Frames, concepts, and conceptual fields. In Lehrer, A. and Kittay, E., editors, *Frames, Fields, and Contrasts: New Essays in Semantic and Lexical Organisation*, pages 21–74. Lawrence Erlbaum, Hillsdale, NJ.
- [Barsalou and Sewell, 1985] Barsalou, L. and Sewell, D. (1985). Contrasting the representation of scripts and categories. *Journal of Memory and Language*, 24(6):646–665.
- [Bartlett, 1932] Bartlett, F. (1932). *Remembering*. Cambridge University Press, Cambridge.
- [Bartsch, 1987] Bartsch, R. (1987). Frame representations and discourse representations. *Theoretical Linguistics*, 14:65–117.
- [Bobrow, 1968] Bobrow, D. (1968). Natural language input for a computer problem-solving system. In Minsky, M., editor, *Semantic Information Processing*, pages 133–215. MIT Press, Cambridge, MA.
- [Bonitatibus and Beal, 1996] Bonitatibus, G. and Beal, C. (1996). Finding new meanings: Children’s recognition of interpretive ambiguity in text. *Journal of Experimental Child Psychology*, 62(1):131–150.
- [Bower et al., 1979] Bower, G., Black, J., and Turner, T. (1979). Scripts in memory for text. *Cognitive Psychology*, 11(2):177–220.
- [Brewka et al., 1997] Brewka, G., Dix, J., and Konolige, K. (1997). *Nonmonotonic Reasoning: an Overview*. Centre for the Study of Language and Information Lecture Notes 73, Stanford, California.
- [Broadbent, 1958] Broadbent, D. (1958). *Perception and Communication*. Pergamon Press, London.
- [Carroll, 1998] Carroll, R. (1998). *The Skeptic’s Dictionary*. Web page: <http://skepdic.com/>.

- [Charniak, 1972] Charniak, E. (1972). Toward a model of children's story comprehension. Technical Report AI-TR-266, MIT AI Lab, Cambridge, MA.
- [Charniak, 1983] Charniak, E. (1983). Passing markers: A theory of contextual influence in language comprehension. *Cognitive Science*, 7(3):171–190.
- [Charniak, 1986] Charniak, E. (1986). A neat theory of marker passing. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, volume 1, pages 584–588, Philadelphia.
- [Charniak and Goldman, 1989] Charniak, E. and Goldman, R. (1989). A semantics for probabilistic quantifier-free first-order languages, with particular application to story understanding. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 1074–1079. Detroit.
- [Charniak and Goldman, 1990] Charniak, E. and Goldman, R. (1990). Plan recognition in stories and in life. In Henrion, M., Shachter, R. D., Kanal, L. N., and Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence 5*, volume 10 of *Machine Intelligence and Pattern Recognition*, pages 343–352. North-Holland, Amsterdam.
- [Charniak and Goldman, 1993] Charniak, E. and Goldman, R. (1993). A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79.
- [Charniak and McDermott, 1985] Charniak, E. and McDermott, D. (1985). *Introduction to Artificial Intelligence*. Addison-Wesley, Reading, MA.
- [Charniak and Shimony, 1990] Charniak, E. and Shimony, S. (1990). Probabilistic semantics for cost-based abduction. Technical Report CS-90-02, Department of Computer Science, Brown University.
- [Charniak and Shimony, 1994] Charniak, E. and Shimony, S. (1994). Cost-based abduction and MAP explanation. *Artificial Intelligence*, 66(2):345–374.
- [Clancey, 1983] Clancey, W. (1983). The advantages of abstract control knowledge in expert system design. In *Proceedings of the 2nd National Conference on Artificial Intelligence (AAAI-83)*, pages 74–78, Washington, DC.
- [Collins English Dictionary, 1994] Collins English Dictionary (1994).

- [Cooper and Shallice, 2000] Cooper, R. and Shallice, T. (2000). Contention scheduling and the control of routine activities. *Cognitive Neuropsychology*, 17(4):297–338.
- [Cooper et al., 1995] Cooper, R., Shallice, T., and Farrington, J. (1995). Symbolic and continuous processes in the automatic selection of actions. In Hallam, J., editor, *Hybrid Problems, Hybrid Solutions*, pages 27–37. IOS Press, Amsterdam.
- [Copi, 1978] Copi, I. (1978). *Introduction to Logic*. MacMillan, New York, NY, 5th edition.
- [Correia, 1980] Correia, A. (1980). Computing story trees. *American Journal of Computational Linguistics*, 6(3-4):135–149.
- [Crain and Steedman, 1985] Crain, S. and Steedman, M. (1985). On not being led up the garden path: The use of context by the psychological syntax processor. In Dowty, D., Karttunen, L., and Zwicky, A., editors, *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*, pages 320–358. Cambridge University Press.
- [Cycorp, 2000] Cycorp (2000). Publicity material. Web page: <http://www.cyc.com/>.
- [Dahlgren, 1988] Dahlgren, K. (1988). *Naive Semantics for Natural Language Understanding*. Kluwer Academic Publishers, Norwell, MA.
- [Dalrymple, 1988] Dalrymple, M. (1988). The interpretation of tense and aspect in english. In *Proceedings of 26th Meeting of the Association for Computational Linguistics (ACL-88)*, pages 68–74, Buffalo, NY.
- [Davis, 1980] Davis, R. (1980). Meta-rules: Reasoning about control. *Artificial Intelligence*, 15(3):179–222.
- [de Kleer, 1986] de Kleer, J. (1986). An assumption-based truth maintenance system. *Artificial Intelligence*, 28(2):127–162.
- [DeJong, 1979] DeJong, G. (1979). Prediction and substantiation: A new approach to natural language processing. *Cognitive Science*, 3(3):251–273.
- [Doyle, 1979] Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence*, 12(3):231–272.

- [Dretske, 1981] Dretske, F. (1981). *Knowledge and the Flow of Information*. Blackwell, Oxford.
- [Dyer et al., 1992] Dyer, M., Cullingford, R., and Alvarado, S. (1992). Scripts. In Shapiro, S., editor, *Encyclopaedia of Artificial Intelligence*. John Wiley and Sons, 2nd edition.
- [Eberle, 1992] Eberle, K. (1992). On representing the temporal structure of a natural language text. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING-92)*, volume 2, pages 288–294, Nantes, France.
- [Eliot, 1958] Eliot, T. (1958). *Collected Poems 1909-1935*. Faber and Faber, London.
- [Enkvist, 1989] Enkvist, N. (1989). Connexity, interpretability, universes of discourse, and text worlds. In Allen, S., editor, *Possible Worlds in Humanities, Arts and Sciences (Proceedings of Nobel Symposium 65)*, pages 162–186. Walter de Gruyter, Berlin.
- [Ericsson and Kintsch, 1995] Ericsson, K. and Kintsch, W. (1995). Long-term working memory. *Psychological Review*, 102(2):211–245.
- [Eysenck and Keane, 1995] Eysenck, M. and Keane, M. (1995). *Cognitive Psychology: A Student's Handbook*. Psychology Press, Hove, UK, third edition.
- [Fillmore, 1968] Fillmore, C. (1968). The case for case. In Bach, E. and Harms, R., editors, *Universals in Linguistic Theory*, pages 1–88. Holt, Rinehart and Winston, New York, NY.
- [Fletcher et al., 1990] Fletcher, C., Hummel, J., and Marsolek, C. (1990). Causality and the allocation of attention during comprehension. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 16(2):233–240.
- [Fletcher et al., 1996] Fletcher, C., van den Broek, P., and Arthur, E. (1996). A model of narrative comprehension and recall. In Britton, B. and Graesser, A., editors, *Models of Understanding Text*, pages 141–163. Lawrence Erlbaum, Mahwah, NJ.
- [Forbus and Oblinger, 1990] Forbus, K. and Oblinger, D. (1990). Making SME greedy and pragmatic. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society*, pages 61–68, Cambridge, MA.
- [Frazier and Fodor, 1978] Frazier, L. and Fodor, J. (1978). The sausage machine: A new two-stage parsing model. *Cognition*, 6:417–459.

- [Frith, 1989] Frith, U. (1989). *Autism: Explaining the Enigma*. Blackwell, Cambridge, MA.
- [Frost, 1986] Frost, R. (1986). *Introduction to Knowledge Base Systems*. Collins, London, UK.
- [Gardner, 1987] Gardner, H. (1987). *The Mind's New Science: A History of the Cognitive Revolution*. Basic Books, USA.
- [Garnham, 1996] Garnham, A. (1996). Discourse comprehension models. In Dijkstra, T. and de Smedt, K., editors, *Computational Psycholinguistics*, pages 221–244. Taylor and Francis.
- [Gathercole and Baddeley, 1993] Gathercole, S. and Baddeley, A. (1993). *Working Memory and Language*. Lawrence Erlbaum, Hove, UK.
- [Georgeff, 1982] Georgeff, M. (1982). Procedural control in production systems. *Artificial Intelligence*, 18(2):175–201.
- [Gernsbacher et al., 1990] Gernsbacher, M., Varner, K., and Faust, M. (1990). Investigating differences in general comprehension skill. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 16(3):430–445.
- [Goldman, 1990] Goldman, R. (1990). *A Probabilistic Approach to Language Understanding*. PhD thesis, Department of Computer Science.
- [Goldszmidt and Pearl, 1996] Goldszmidt, M. and Pearl, J. (1996). Qualitative probabilities for default reasoning, belief revision, and causal modeling. *Artificial Intelligence*, 84(1–2):57–112.
- [Graesser et al., 1994] Graesser, A., Singer, M., and Trabasso, T. (1994). Constructing inferences during narrative text comprehension. *Psychological Review*, 101(3):371–395.
- [Graesser et al., 1996] Graesser, A., Swamer, S., Baggett, W., and Sell, M. (1996). New models of deep comprehension. In Britton, B. and Graesser, A., editors, *Models of Understanding Text*, pages 1–32. Lawrence Erlbaum, Mahwah, NJ.
- [Graesser et al., 1997] Graesser, A., Swamer, S., and Hu, X. (1997). Quantitative discourse psychology. *Discourse Processes*, 23(3):229–263.

- [Grice, 1975] Grice, H. (1975). Logic and conversation. In Cole, P. and Morgan, J., editors, *Syntax and Semantics 3: Speech Acts*, pages 41–58. Academic Press, New York, NY.
- [Haberlandt and Bingham, 1984] Haberlandt, K. and Bingham, G. (1984). The effect of input direction on the processing of script statements. *Journal of Verbal Learning and Verbal Behaviour*, 23:162–177.
- [Haberlandt and Graesser, 1990] Haberlandt, K. and Graesser, A. (1990). Integration and buffering of new information. *The Psychology of Learning and Motivation*, 25:71–87.
- [Hayes, 1980] Hayes, P. (1980). The logic of frames. In Metzing, D., editor, *Frame Conceptions and Text Understanding*, pages 46–61. Walter de Gruyter, Berlin.
- [Hayes-Roth, 1985] Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321.
- [Hempel, 1966] Hempel, C. (1966). *Philosophy of Natural Science*. Prentice-Hall, Englewood Cliffs, NJ.
- [Hintzman, 1986] Hintzman, D. (1986). “Schema abstraction” in a multiple-trace memory model. *Psychological Review*, 93(4):411–428.
- [Hoadley et al., 1994] Hoadley, C., Ranney, M., and Schank, P. (1994). WanderECHO: A connectionist simulation of limited coherence. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pages 421–426, Georgia Institute of Technology.
- [Hobbs, 1979] Hobbs, J. (1979). Coherence and coreference. *Cognitive Science*, 3(1):67–90.
- [Hobbs, 1990] Hobbs, J. (1990). *Literature and Cognition*. CSLI Lecture Notes 21. Center for the Study of Language and Information.
- [Hobbs et al., 1993] Hobbs, J., Stickel, M., Appelt, D., and Martin, P. (1993). Interpretation as abduction. *Artificial Intelligence*, 63(1–2):69–142.
- [Holyoak, 1991] Holyoak, K. (1991). Symbolic connectionism: Toward third-generation theories of expertise. In Ericsson, K. and Smith, J., editors, *Towards a General Theory of Expertise: Prospects and Limits*. Cambridge University Press, New York, NY.

- [Hongua, 1994] Hongua, G. (1994). Understanding a story with causal relationships. In *Proceedings of the 8th International Symposium on Methodologies for Intelligent Systems (ISMIS-94)*, pages 265–274, Charlotte, North Carolina.
- [Ingarden, 1973] Ingarden, R. (1931/1973). *The Literary Work of Art: An Investigation on the Borderlines of Ontology, Logic, and Theory of Literature (English translation, from 1931 Polish edition)*. Northwestern University Press, Evanston, ILL.
- [Iser, 1971] Iser, W. (1971). The reading process: A phenomenological approach. *New Literary History*, 3(2):279–299.
- [Jacobs, 1988] Jacobs, P. (1988). Concretion: Assumption-based understanding. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING-88)*, volume 1, pages 270–274, Budapest.
- [Johnson and Seifert, 1999] Johnson, H. and Seifert, C. (1999). Modifying mental representations: Comprehending corrections. In Goldman, S. and van Oostendorp, H., editors, *The Construction of Mental Representations During Reading*, pages 303–318. Lawrence Erlbaum, Mahwah, NJ.
- [Johnson-Laird, 1988] Johnson-Laird, P. (1988). Freedom and constraint in creativity. In Sternberg, R., editor, *The Nature of Creativity*. Cambridge University Press, Cambridge, UK.
- [Jonides, 1995] Jonides, J. (1995). Working memory and thinking. In Smith, E. and Osherson, D., editors, *An Invitation to Cognitive Science*, volume 3: Thinking. MIT Press, Cambridge, MA.
- [Josephson and Josephson, 1996] Josephson, J. and Josephson, S. (1996). *Abductive Inference: Computation, Philosophy, Technology*. Cambridge University Press, New York, NY.
- [Just and Carpenter, 1992] Just, M. and Carpenter, P. (1992). A capacity theory of comprehension: Individual differences in working memory. *Psychological Review*, 99(1):122–149.
- [Kamas et al., 1996] Kamas, E., Reder, L., and Ayers, M. (1996). Partial matching in the mooses illusion: Response bias not sensitivity. *Memory and Cognition*, 24(6):687–699.

- [Kamp and Reyle, 1993] Kamp, H. and Reyle, U. (1993). *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer Academic Press, Dordrecht.
- [Kaplan and Berry-Rogghe, 1991] Kaplan, R. and Berry-Rogghe, G. (1991). Knowledge-based acquisition of causal relationships in text. *Knowledge Acquisition*, 3:317–337.
- [Kautz, 1990] Kautz, H. (1990). A circumscriptive theory of plan recognition. In Cohen, P., Morgan, J., and Pollack, M., editors, *Intentions in Communication*, pages 105–133. MIT Press, Cambridge, MA.
- [Kautz and Allen, 1986] Kautz, H. and Allen, J. (1986). Generalized plan recognition. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*, volume 1, pages 32–37, Philadelphia.
- [Kayser and Coulon, 1981] Kayser, D. and Coulon, D. (1981). Variable-depth natural language understanding. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI-81)*, volume 1, pages 64–66, University of British Columbia, Vancouver, Canada.
- [Keefe and McDaniel, 1993] Keefe, D. and McDaniel, M. (1993). The time course and durability of predictive inferences. *Journal of Memory and Language*, 32(4):446–463.
- [Kehler, 1995] Kehler, A. (1995). *Interpreting Cohesive Forms in the Context of Discourse Inference*. PhD thesis, Harvard University.
- [Kelley, 1973] Kelley, H. (1973). The processes of causal attribution. *American Psychologist*, 28(2):107–128.
- [Kintsch, 1988] Kintsch, W. (1988). The role of knowledge in discourse comprehension: A construction-integration model. *Psychological Review*, 95(2):163–182.
- [Kintsch, 1998] Kintsch, W. (1998). *Comprehension: A Paradigm for Cognition*. Cambridge University Press, Cambridge.
- [Kintsch and Mannes, 1987] Kintsch, W. and Mannes, S. (1987). Generating scripts from memory. In van der Meer, E. and Hoffmann, J., editors, *Knowledge Aided Information Processing*, pages 61–80. Elsevier Science Publishers, North Holland.

- [Kintsch and van Dijk, 1978] Kintsch, W. and van Dijk, T. (1978). Toward a model of text comprehension and production. *Psychological Review*, 85(5):363–394.
- [Kolodner, 1992] Kolodner, J. (1992). An introduction to case-based reasoning. *Artificial Intelligence Review*, 6(1):3–34.
- [Krems and Johnson, 1995] Krems, J. and Johnson, T. (1995). Integration of anomalous data in multicausal explanations. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, pages 277–282, Pittsburgh.
- [Laird et al., 1987] Laird, J., Newell, A., and Rosenbloom, P. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64.
- [Leake, 1992] Leake, D. (1992). Using goals and experience to guide abduction. Technical Report TR359, Computer Science Department, Indiana University.
- [Lehnert et al., 1983] Lehnert, W., Dyer, M., Johnson, P., Yang, C., and Harley, S. (1983). BORIS - an experiment in in-depth understanding of narratives. *Artificial Intelligence*, 20(1):15–62.
- [Lindsay and Norman, 1977] Lindsay, P. and Norman, D. (1977). *Human Information Processing*. Academic Press, New York, NY.
- [Long et al., 1990] Long, D., Golding, J., Graesser, A., and Clark, L. (1990). Goal, event and state inferences: An investigation of inference generation during story comprehension. *The Psychology of Learning and Motivation*, 25:89–102.
- [Long et al., 1996] Long, D., Seely, M., Oppy, B., and Golding, J. (1996). The role of inferential processing in reading ability. In Britton, B. and Graesser, A., editors, *Models of Understanding Text*, pages 189–214. Lawrence Erlbaum, Mahwah, NJ.
- [van Oostendorp and Bonebakker, 1999] van Oostendorp, H. and Bonebakker, C. (1999). Difficulties in updating mental representations during reading. In Goldman, S. and van Oostendorp, H., editors, *The Construction of Mental Representations During Reading*, pages 319–339. Lawrence Erlbaum, Mahwah, NJ.
- [McCarthy, 1980] McCarthy, J. (1980). Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1–2):27–39.

- [McKoon and Ratcliff, 1979] McKoon, G. and Ratcliff, R. (1979). Priming in episodic and semantic memory. *Journal of Verbal Learning and Verbal Behaviour*, 18(4):463–480.
- [McKoon and Ratcliff, 1992] McKoon, G. and Ratcliff, R. (1992). Inference during reading. *Psychological Review*, 99(3):440–466.
- [McKoon et al., 1986] McKoon, G., Ratcliff, R., and Dell, G. (1986). A critical evaluation of the semantic-episodic distinction. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 12(2):295–306.
- [Miller, 1956] Miller, G. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(1):81–97.
- [Minsky, 1975] Minsky, M. (1975). A framework for representing knowledge. In Winston, P., editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York, NY.
- [Moens and Steedman, 1988] Moens, M. and Steedman, M. (1988). Temporal ontology and temporal reference. *Computational Linguistics*, 14(2):15–28.
- [Mooney, 1990] Mooney, R. (1990). Learning plan schemata from observation: Explanation-based learning for plan recognition. *Cognitive Science*, 14(4):483–509.
- [Moorman, 1997] Moorman, K. (1997). *A Functional Theory of Creative Reading: Process, Knowledge, and Evaluation*. PhD thesis, College of Computing.
- [Murray et al., 1993] Murray, J., Klin, C., and Myers, J. (1993). Forward inferences in narrative text. *Journal of Memory and Language*, 32:464–473.
- [Myers and O'Brien, 1998] Myers, J. and O'Brien, E. (1998). Accessing the discourse representation during reading. *Discourse Processes*, 26(2–3):131–157.
- [Nagao, 1993] Nagao, K. (1993). Abduction and dynamic preference in plan-based dialogue understanding. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, volume 2, pages 1186–1192, San Mateo, CA.
- [Newell and Simon, 1972] Newell, A. and Simon, H. (1972). *Human Problem Solving*. Prentice-Hall, Englewood Cliffs.

- [Ng, 1992] Ng, H. (1992). *A General Abductive System with Application to Plan Recognition and Diagnosis*. PhD thesis, Artificial Intelligence Laboratory, University of Texas at Austin.
- [Ng and Mooney, 1989] Ng, H. and Mooney, R. (1989). Abductive explanation in text understanding: Some problems and solutions. Technical Report AI89-116, University of Texas at Austin.
- [Ng and Mooney, 1990] Ng, H. and Mooney, R. (1990). On the role of coherence in abductive explanation. In *Proceedings of the 8th AAAI*, pages 337–342, Boston, MA.
- [Ng and Mooney, 1992] Ng, H. and Mooney, R. (1992). A first-order abductive system and its use in plan recognition and diagnosis. Unpublished technical report, University of Texas at Austin.
- [Noordman and Vonk, 1992] Noordman, L. and Vonk, W. (1992). Reader's knowledge and the control of inferences in reading. *Language and Cognitive Processes*, 7(3-4):373–391.
- [Noordman and Vonk, 1998] Noordman, L. and Vonk, W. (1998). Memory-based processing in understanding causal information. *Discourse Processes*, 26(2–3):191–212.
- [Norvig, 1989] Norvig, P. (1989). Marker passing as a weak method for text inferencing. *Cognitive Science*, 13(4):569–620.
- [Norvig and Wilensky, 1990] Norvig, P. and Wilensky, R. (1990). A critical evaluation of commensurable abduction models for semantic interpretation. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90)*, volume 3, pages 224–230, Helsinki.
- [Nowak and Thagard, 1992] Nowak, G. and Thagard, P. (1992). Copernicus, Ptolemy, and explanatory coherence. In Giere, R., editor, *Cognitive Models of Science*, pages 274–309. University of Minnesota Press, Minneapolis.
- [O'Brien et al., 1988] O'Brien, E., Shank, D., Myers, J., and Rayner, K. (1988). Elaborative inferences during reading: Do they occur on-line? *Journal of Experimental Psychology: Learning, Memory and Cognition*, 14(3):410–420.

- [Ozonoff et al., 1991] Ozonoff, S., Pennington, B. F., and Rogers, S. J. (1991). Executive function deficits in high functioning autistic individuals: Relationship to theory of mind. *Journal of Child Psychology and Psychiatry*, 32(7):1081–1105.
- [Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, CA.
- [Peng and Reggia, 1990] Peng, Y. and Reggia, R. (1990). *Abductive Inference Models for Diagnostic Problem Solving*. Springer-Verlag, New York, NY.
- [Pereira, 1985] Pereira, F. (1985). A new characterization of attachment preferences. In Dowty, D., Karttunen, L., and Zwicky, A., editors, *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*, pages 307–319. Cambridge University Press.
- [Quillian, 1968] Quillian, M. (1968). Semantic memory. In M. Minsky, editor, *Semantic Information Processing*, pages 216–270. MIT Press, Cambridge, MA.
- [Ram, 1991] Ram, A. (1991). A theory of questions and question asking. *The Journal of the Learning Sciences*, 1(3-4):273–318.
- [Read, 1987] Read, S. (1987). Constructing causal scenarios: A knowledge structure approach to causal reasoning. *Journal of Personality and Social Psychology*, 52(2):288–302.
- [Read and Marcus-Newhall, 1993] Read, S. and Marcus-Newhall, A. (1993). Explanatory coherence in social explanations: A parallel distributed processing account. *Journal of Personality and Social Psychology*, 65(3):429–447.
- [Reichgelt, 1989] Reichgelt, H. (1989). *Knowledge Representation: An AI Perspective*. Ablex Publishing Corporation, Norwood, NJ.
- [Reiter, 1980] Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132.
- [Rich and Knight, 1991] Rich, E. and Knight, K. (1991). *Artificial Intelligence*. McGraw-Hill, second edition.
- [Richter, 1965] Richter, H. (1965). *Dada: Art and Anti-Art*. Thames and Hudson, London.

- [Robbe-Grillet, 1959] Robbe-Grillet, A. (1959). *Jealousy*. John Calder, London.
- [Ronen, 1994] Ronen, R. (1994). *Possible Worlds in Literary Theory*. Cambridge University Press, Cambridge, UK.
- [Rosch, 1978] Rosch, E. (1978). Principles of categorisation. In Rosch, E. and Lloyd, B., editors, *Cognition and Categorisation*, pages 27–48. Lawrence Erlbaum, Hillsdale, NJ.
- [Rumelhart, 1975] Rumelhart, D. (1975). Notes on a schema for stories. In Bobrow, D. and Collins, A., editors, *Representation and Understanding*, pages 211–236. Academic Press, New York, NY.
- [Ryan, 1985] Ryan, M.-L. (1985). The modal structure of narrative universes. *Poetics Today*, 6(4):717–755.
- [Sanford and Garrod, 1981] Sanford, A. and Garrod, S. (1981). *Understanding Written Language: Explorations of Comprehension Beyond the Sentence*. John Wiley and Sons.
- [Schank, 1975] Schank, R. (1975). SAM: a story understander. Technical Report 43, Department of Computer Science, Yale University.
- [Schank, 1978] Schank, R. (1978). Predictive understanding. In Campbell, R. and Smith, P., editors, *Recent Advances in the Psychology of Language: Formal and Experimental Approaches*, pages 91–101. Plenum Press, New York, NY.
- [Schank, 1982] Schank, R. (1982). Reminding and memory organisation: An introduction to MOPs. In Lehnert, W. and Ringle, M., editors, *Strategies for Natural Language Processing*, pages 455–493. Lawrence Erlbaum, Hillsdale, NJ.
- [Schank and Abelson, 1977] Schank, R. and Abelson, R. (1977). *Scripts, Plans, Goals and Understanding*. Lawrence Erlbaum, Hillsdale, NJ.
- [Schank and Riesbeck, 1981] Schank, R. and Riesbeck, C. (1981). *Inside Computer Understanding: Five Programs Plus Miniatures*. Lawrence Erlbaum, Hillsdale, NJ.
- [Shallice, 1982] Shallice, T. (1982). Specific impairments of planning. *Philosophical Transactions of the Royal Society of London B*, 298:199–209.
- [Shanahan, 1989] Shanahan, M. (1989). Prediction is deduction but explanation is abduction. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, volume 2, pages 1055–1060, Detroit.

- [Sharkey, 1990] Sharkey, N. (1990). A connectionist model of text comprehension. In Balota, D., d'Arcais, G. F., and Rayner, K., editors, *Comprehension Processes in Reading*. Lawrence Erlbaum, Hillsdale, NJ.
- [Simon and Kadane, 1975] Simon, H. and Kadane, J. (1975). Optimal problem-solving search. *Artificial Intelligence*, 6(3):235–247.
- [Singer, 1994] Singer, M. (1994). Discourse inference processes. In Gernsbacher, M., editor, *Handbook of Psycholinguistics*, pages 479–515. Academic Press, London, UK.
- [Singer et al., 1994] Singer, M., Graesser, A., and Trabasso, T. (1994). Minimal or global inference during reading. *Journal of Memory and Language*, 33(4):421–441.
- [Singer and Halldorson, 1996] Singer, M. and Halldorson, M. (1996). Constructing and validating motive bridging inferences. *Cognitive Psychology*, 30(1):1–38.
- [Smolensky, 1986] Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In McClelland, J., editor, *Parallel Distributed Processing*, volume 1, pages 194–281. MIT Press, Cambridge.
- [Somers, 1992] Somers, H. (1992). Current research in machine translation. In Newton, J., editor, *Computers in Translation: A Practical Appraisal*, pages 189–207. Routledge, London.
- [St. George and Kutas, 1998] St. George, M. and Kutas, M. (1998). Working memory capacity and the uncertainty of predictive inferences. CogSci. UCSD technical report 98.02, University of California, San Diego.
- [Sterling and Shapiro, 1994] Sterling, L. and Shapiro, E. (1994). *The Art of Prolog*. MIT Press, Cambridge, MA.
- [Suh and Trabasso, 1993] Suh, S. and Trabasso, T. (1993). Inferences during reading: Converging evidence from discourse analysis, talk-aloud protocols, and recognition priming. *Journal of Memory and Language*, 32(3):279–301.
- [Thagard, 1988] Thagard, P. (1988). *Computational Philosophy of Science*. MIT Press, Cambridge, MA.
- [Thagard, 1989] Thagard, P. (1989). Explanatory coherence. *Behavioural and Brain Sciences*, 12:435–502.

- [Thagard, 1997] Thagard, P. (1997). Coherent and creative conceptual combinations. In Ward, T., Smith, S., and Viad, J., editors, *Creative thought: An investigation of conceptual structures and processes*, pages 129–141. American Psychological Association, Washington, D.C.
- [Thagard, inpr] Thagard, P. (inpr). Probabilistic networks and explanatory coherence. Technical report, University of Waterloo, web page: <http://cogsci.uwaterloo.ca/Articles/Pages/>
- [Thagard et al., inpr] Thagard, P., Eliasmith, C., Rusnock, P., and Shelley, C. (inpr). Knowledge and coherence. In Elio, R., editor, *Common Sense, Reasoning, and Rationality*, volume 11. Oxford University Press, New York, NY.
- [Thagard and Verbeurgt, 1998] Thagard, P. and Verbeurgt, K. (1998). Coherence as constraint satisfaction. *Cognitive Science*, 22(1):1–24.
- [Thibadeau et al., 1982] Thibadeau, R., Just, M., and Carpenter, P. (1982). A model of the time course and content of reading. *Cognitive Science*, 6(2):157–203.
- [Thorndyke, 1976] Thorndyke, P. (1976). The role of inferences in discourse comprehension. *Journal of Verbal Learning and Verbal Behaviour*, 15(4):437–446.
- [Trabasso and Magliano, 1996] Trabasso, T. and Magliano, J. (1996). Conscious understanding during comprehension. *Discourse Processes*, 21(3):255–287.
- [Trabasso et al., 1995] Trabasso, T., Suh, S., and Payton, P. (1995). Explanatory coherence in understanding and talking about events. In Gernsbacher, M. and Givón, T., editors, *Coherence in Spontaneous Text*, pages 189–214. John Benjamins, Amsterdam.
- [Trabasso and van den Broek, 1985] Trabasso, T. and van den Broek, P. (1985). Causal thinking and the representation of narrative events. *Journal of Memory and Language*, 24:612–630.
- [Trabasso et al., 1989] Trabasso, T., van den Broek, P., and Suh, S. (1989). Logical necessity and transitivity of causal relations in stories. *Discourse Processes*, 12(1):1–25.
- [Tulving, 1985] Tulving, E. (1985). Episodic and semantic memory. In Tulving, E. and Donaldson, W., editors, *Organisation of Memory*, pages 381–403. Academic Press, London.

- [Tulving, 1986] Tulving, E. (1986). What kind of a hypothesis is the distinction between episodic and semantic memory? *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 12(2):307–311.
- [Tzara, 1918] Tzara, T. (1918). Dada manifesto. In Motherwell, R., editor, *Dada Painters and Poets*. George Wittenborn, Inc., New York, NY.
- [van den Broek, 1990a] van den Broek, P. (1990a). The causal inference maker: Towards a process model of inference generation in text comprehension. In Balota, D., d'Arcais, G. F., and Rayner, K., editors, *Comprehension Processes in Reading*. Lawrence Erlbaum, Hillsdale, NJ.
- [van den Broek, 1990b] van den Broek, P. (1990b). Causal inferences and the comprehension of narrative texts. *The Psychology of Learning and Motivation*, 25:175–196.
- [van den Broek, 1994] van den Broek, P. (1994). Comprehension and memory of narrative texts: Inferences and coherence. In Gernsbacher, M., editor, *Handbook of Psycholinguistics*, pages 539–588. Academic Press, London.
- [van den Broek and Lorch, 1993] van den Broek, P. and Lorch, R. (1993). Network representations of causal relations in memory for narrative texts: Evidence from primed recognition. *Discourse Processes*, 16(1–2):75–98.
- [van den Broek et al., 1996] van den Broek, P., Ridsen, K., Fletcher, C., and Thurlow, R. (1996). A “landscape” view of reading: Fluctuating patterns of activation and the construction of a stable memory representation. In Britton, B. and Graesser, A., editors, *Models of Understanding Text*, pages 165–187. Lawrence Erlbaum, Mahwah, NJ.
- [van den Broek et al., 1995] van den Broek, P., Ridsen, K., and Husebye-Hartmann, E. (1995). The role of readers' standards for coherence in the generation of inferences during reading. In R.F. Lorch, J. and O'Brien, E., editors, *Sources of Coherence in Reading*, pages 353–373. Lawrence Erlbaum, Hillsdale, NJ.
- [van den Broek et al., 1999] van den Broek, P., Young, M., Tzeng, Y., and Linderholm, T. (1999). The landscape model of reading: Inferences and the online construction of a memory representation. In Goldman, S. and van Oostendorp, H., editors, *The Construction of Mental Representations During Reading*, pages 165–187. Lawrence Erlbaum, Mahwah, NJ.

- [van Dijk, 1977] van Dijk, T. (1977). Semantic macro-structures and knowledge frames in discourse comprehension. In Just, M. and Carpenter, P., editors, *Cognitive Processes in Comprehension*, pages 3–32. Lawrence Erlbaum, Hillsdale, NJ.
- [van Fraassen, 1977] van Fraassen, B. (1977). The pragmatics of explanation. *American Philosophical Quarterly*, 14:143–150.
- [Walker and Yekovich, 1984] Walker, C. and Yekovich, F. (1984). Script-based inferences: Effects of text and knowledge variables on recognition memory. *Journal of Verbal Learning and Verbal Behaviour*, 23(3):357–370.
- [Whitney et al., 1991] Whitney, P., Ritchie, B., and Clark, M. (1991). Working-memory capacity and the use of elaborative inferences in text comprehension. *Discourse Processes*, 14(2):133–145.
- [Wilensky, 1983] Wilensky, R. (1983). Memory and inference. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-83)*, volume 1, pages 402–404, Karlsruhe, West Germany.
- [Wilensky, 1992] Wilensky, R. (1992). Discourse versus probability in the theory of natural language interpretation. In *Proceedings of the AAAI Symposium on Probability and Natural Language Processing*, pages 128–135, Cambridge, MA.
- [Wilensky et al., 1988] Wilensky, R., Chin, D., Luria, M., Martin, J., Mayfield, J., and Dekai, W. (1988). The Berkeley UNIX consultant project. *Computational Linguistics*, 14(4):35–84.
- [Williams, 1992] Williams, C. (1992). *ATLAS: A Natural Language Understanding System*. PhD thesis, Department of Computer Science.
- [Zwaan, 1996] Zwaan, R. (1996). Toward a model of literary comprehension. In Britton, B. and Graesser, A., editors, *Models of Understanding Text*, pages 241–255. Lawrence Erlbaum, Hillsdale, NJ.
- [Zwaan and Graesser, 1993] Zwaan, R. and Graesser, A. (1993). Reading goals and situation models. *Psychology (web-based)*, 4(3).
- [Zwaan and Van Oostendoorp, 1993] Zwaan, R. and Van Oostendoorp, H. (1993). Do readers construct spatial representations in naturalistic story comprehension? *Discourse Processes*, 16(1–2):125–143.

Where *inpr* appears as the year of publication, the item was in press at the time of writing (July 2000).