UNIVERSITY
of
GLASGOW

Department of
Computing Science

Analysing Accident Reports Using Structured and Formal Methods

Colin Paul Burns

Submitted for the degree of Doctor of Philosophy
to
The University of Glasgow,
February, 2000.

Abstract

Formal methods are proposed as a means to improve accident reports, such as the report into the 1996 fire in the Channel Tunnel between the UK and France. The size and complexity of accident reports create difficulties for formal methods, which traditionally suffer from problems of scalability and poor readability. This thesis demonstrates that features of an engineering-style formal modelling process, particularly the structuring of activity and management of information, reduce the impact of these problems and improve the accuracy of formal models of accident reports. This thesis also contributes a detailed analysis of the methodological requirements for constructing accident report models.

Structured, methodical construction and mathematical analysis of the models elicits significant problems in the content and argumentation of the reports. Once elicited, these problems can be addressed. This thesis demonstrates the benefits and limitations of taking a wider scope in the modelling process than is commonly adopted for formal accident analysis.

We present a deontic action logic as a language for constructing models of accident reports. Deontic action models offer a novel view of the report, which highlights both the expected and actual behaviour in the report, and facilitates examination of the conflict between the two. This thesis contributes an objective analysis of the utility of both deontic and action logic operators to the application of modelling accident reports. A tool is also presented that executes a subset of the logic, including these deontic and action logic operators.

Contents

List of Figures

Acknowledgements

I would like to acknowledge my debt to the many members of the Department of Computing Science at the University of Glasgow who have assisted, directly or indirectly, in the research that led to this thesis. In particular, my supervisors, Professor Chris Johnson and Professor Muffy Calder, have both provided me with generous levels of guidance and support throughout the course of this work. The work described has also benefited greatly from the diverse views of my fellow members (past and present) of the Glasgow Accident Analysis Group.

Professor Anthony Finkelstein contributed ideas and hard to find literature that helped greatly in keeping this thesis on track. I would also like to thank Dave Robertson, from the Division of Informatics at the University of Edinburgh, for his freely given and invaluable advice on the development of interpreters in Prolog.

Finally, I must express my gratitude to Amanda McLeod and my parents for their support and patience over the last few years.

This work was funded by an EPSRC studentship.

Declaration

This thesis was composed by myself and the work it contains is my own.

Part I

Introduction

Chapter 1

Motivation and Background

This thesis demonstrates that using formal methods can improve the accuracy of key information in accident reports. Accident reports are produced from inquiries into major accidents and the findings can have direct implications on the design of future systems. It is therefore imperative that the findings are presented clearly and accurately to the users of the report. Formal methods are commonly used in safety-critical design, where inaccuracy can have extremely serious consequences. Omitted, ambiguous, or inaccurate information in a report can lead to unsafe system designs and misdirected legislation [Lev95, BR91]. We demonstrate that using formal methods to analyse accident reports facilitates the identification of errors and omissions in the report, and the verification of the argumentation.

Previous work in this domain has not addressed the issues of scale involved with modelling accident reports. We use a structured requirements engineering method, Structured Common Sense (SCS) [PFAB86], to guide the elicitation of detail from the report and the construction of the formal model. The term *structured method* is used to refer to a method that guides and structures the activity of the analyst. We demonstrate the significant benefits gained from using such a method. The information in the formal model should be able to be reviewed by non-logicians, and to this end a tool was developed for executing behaviour traces.

This thesis presents a novel approach to formal accident report modelling. The full report is semi-formally modelled before the scope is narrowed for the formal model. We demonstrate that this approach is feasible and discuss its potential benefits, which include reduced confirmation bias and improved accuracy of the formal model.

We present the novel application of a deontic action logic, Extended Deontic Action Logic (EDAL), to the task of accident report analysis. We demonstrate the relevance of the deontic and action-based operators to accident analysis, and we examine the suitability of EDAL itself, for constructing and analysing accident report models.

We employ the SCS method and EDAL language in modelling real accident reports.

This facilitates the study of issues of complexity and scale in the modelling process. It also increases confidence in the validity of our findings with respect to an industrial application of the process.

This work is part of the Glasgow Accident Analysis Group's exploration of existing accident reporting mechanisms and of potential means to improve these mechanisms.

The work described in this thesis is based on two diverse domains. The 'accident analysis' domain draws on safety-critical systems design (such as Leveson [Lev95]) and the study of human error (such as Hollnagel [Hol93]). The 'deontic action logic' domain draws on formal languages and methods, particularly deontic and action-based languages (such as Deontic Action Logic [Kho88]). The following sections introduce the relevant areas in these domains and motivate the need for this work.

1.1 Accident Analysis

> Experience is the best of schoolmasters, only the school fees are heavy.
> *Thomas Carlyle, [Car47]*

Accidents, of differing degrees of severity, will always occur. The harsh consequences of accidents occurring in certain systems, such as nuclear power plants, ensure safety is a primary concern in the design of such systems. Engineers, designers, and other decision makers use information on past failures to inform future designs and decisions [Car89]. However, there are a growing number of cases where problems identified with one system are uncorrected and lead to accidents in other similar systems, such as the Herald of Free Enterprise and the Estonia ferry disasters [Wat97]. Improvements to the accuracy and presentation of the information on past failures will help reduce accident recurrence [Lev95].

In this thesis, we use the report from the fire in the Channel Tunnel linking the UK and France in 1996 [All97] to build a deontic action-based model. The construction and analysis of this model reveal a number of deficiencies in the report.

1.1.1 What is an Accident?

Concerns with safety are addressed in many distinct fields, a factor that has led to numerous and inconsistent definitions for even the most fundamental concepts. To reduce the risk of ambiguity, definitions of some of the important accident analysis and accident modelling terms are presented here. A glossary of the main terms used in this thesis can be found in Appendix A.

A *failure* is the 'non-performance or inability of the system or component to perform its intended function for a specified time under specified environmental conditions' [Lev95]. An *action*, or event, is a 'mechanism through which systems exhibit change' [Kho88]. An *accident* is defined as 'an undesired and unplanned (but not necessarily

unexpected) event that results in (at least) a specified level of loss'. An *incident* is an action 'that involves no loss (or only minor loss) but with potential for loss under different circumstances' [Lev95]. The accidents we examine in this document occur in *safety-critical systems*. Isaksen et al. define these as systems 'whose incorrect function (failure) may have very serious consequences such as loss of human life, severe injuries, large scale environmental damage, or considerable economical penalties' [IBN96]. A system can be described as *safe* if it is free from accidents and loss [Lev95]. A *hazard* is a set of conditions that, in conjunction with other conditions, will lead to an accident [Lev95]. *Safety engineering* is concerned with eliminating hazards or reducing their probability or severity [Lev95].

1.1.2 Why do Accidents Occur and Recur?

Major advances in safety have been made. For example, Nagel [Nag88] reports commercial aircraft accident rates (where an accident is defined as a fatality or a hull loss) dropping from forty-five per million departures in 1960 to two per million departures in 1982. Few accidents are now caused by a single error or failure; instead they generally stem from complex sets of technical, organisational, sociological, and environmental factors, some of which may have existed in the system for many years, which combine in an unforeseen way to penetrate the defences of the system [Lev95, Rea97]. The term *organisational accident* is used to describe such accidents [Rea97].

'Human error' and the problems caused by introduction of 'new technology' are commonly blamed for causing accidents. Senders and Moray [SM91] define *human error* as 'a deviation from expected human behaviour'. Hollnagel [Hol93], Rasmussen [Ras89] and Reason [Rea90] have all identified human error and operator mismanagement as critical factors in the lead up to major accidents. Lekberg [Lek97] states that 'human behaviour is always in one way or another involved in the cause of an accident'. However, the flexibility of human operators prevents more accidents than it causes [Rea97] and Norman [Nor90] blames the seeming ubiquity of human error 'causes' on the misattribution of design errors. Mellor [Mel94] agrees, stating that operator error should be the starting point of investigations rather than the conclusion. Maurino [Mau97] argues that errors in human behaviour must be taken for granted, particularly in stressful situations and systems must be designed to be more tolerant.

The role of new technology, such as computer systems, in accidents is less subjective and contentious. The Ariane 5 Rocket crash and the Therac-25 radiation machine overdosing incidents are well documented examples. Kletz [Kle88] argues that new technology does not introduce new error forms, rather it enables increased process complexity. Mellor [Mel94] and Neumann [Neu95] provide many examples of accidents resulting from increased complexity and coupling in computer systems

Human error and new technology failures are often symptomatic of poor design and management, both of which are general engineering concerns [Lev95]. Rather than focus on the behaviour of specific aspects of the system, the analysis presented here views the system in which an accident occurs as a *total system*; mechanical and human

entities are treated equally as system components. This approach enables us to examine the entire system rather than being restricted to specific components. Furthermore, this approach also enables the analyst to model a wide range of accidents and different levels of granularity within accidents (from interactions between teams of people to system-system interactions). Separating or discriminating between the groups of entities, such as by modelling only human errors, could lead to misattribution of blame and confusion or loss of entity interaction. Indeed, neither humans nor computers are particularly unsafe in themselves, only in their contribution to the full system context [Lev95].

1.1.3 Safety Engineering

The discipline of engineering relies on the application of 'time-tested techniques', combined with analysis of failed designs [Car89, LJ82]. George Stephenson declared that 'a faithful account of those accidents, and of the means by which the consequences were met, was really more valuable than a description of the most successful works.' (quoted in [Pet85]).

In many industries, particularly safety-critical industries, the incidence rate of accidents is decreasing and the potential cost of accidents is increasing. Toft and Reynolds [TR94] recommend the adoption of a proactive policy of examining incidents, both within and outside the industry, for relevant aspects.

Similar accidents occur in disparate organisations where the similarities between the scenarios are not immediately obvious. For example, both the Three Mile Island (nuclear) and Kegworth (aeronautical) accidents exhibit operator confusion between left and right and failure to heed instrumentation feedback [JMW95, Lev95]. However, the difficulty in making 'cross-context comparisons of complex organisations' has been well documented [Joh76]. Worryingly, accidents occur in scenarios with an obvious similarity and where there is no apparent reason why recommendations went unheeded. For example, the King's Cross Underground escalator fire that occurred London in 1987 was ignited by smokers' materials. In the 30 years prior to this fire, there were 46 reported escalator fires in London Underground and over 30 were attributed to smokers' materials. Fennell [Fen88] comments on page 117 of the report:

> London Underground's failure to carry through the proposals resulting from earlier fires - such as the provision of automatic sprinklers, the need to ensure all fire equipment was correctly positioned and serviceable, identification of alternative means of escape, and the need to train staff to react properly and positively in emergencies - was a failure which I believe contributed to the disaster at King's Cross.

The recurrence of accidents and the occurrence of similar accidents implies that problems exist with the dissemination and uptake of the findings of accident inquiries. Although poor safety engineering or a poor safety culture in an industry is often blamed, significant problems have been identified with the main means of this dissemination, the accident report. This thesis addresses these problems.

1.1.4 Accident Reports

An accident report is the published results of an investigation into an accident. The investigation is performed either internally, by the organisation involved, or externally, by a governing body. Investigations can be initiated for a number of reasons. They can be produced as part of litigation or insurance claims to allocate blame or assign responsibility. Criminal investigations use the evidence to link the alleged perpetrators with the crime.

The reports that we focus on are public inquiry reports. These reports are summaries of the germane detail from an in-depth inquiry into the nature and causes, or probable causes, of major accidents, and the inquiry's recommendations to improve the system and prevent a recurrence.

Accident reports are commonly written by the chief accident investigators. They have many different users, including the general public, government officials, and lawyers. However, the stated intention for their use is in improving future systems to prevent a recurrence of the accident. The information in accident reports is used by designers and managers of relevant systems to inform remedial action to reduce the chances of the accident recurring [SJ99]. The work presented here focuses on issues relating to the report being used by designers and managers.

1.1.5 Case Studies

The arguments in this thesis are illustrated with examples drawn predominantly from the public inquiry report produced by the UK Government Department of the Environment Transport and the Regions on the Channel Tunnel fire [All97]. The King's Cross Underground fire report [Fen88] and the North Anna nuclear incident [Dun87] are also used, to a lesser extent. Outlines of these accidents can be found in Appendix B. Although there are some high level similarities between the King's Cross and Channel Tunnel fires, this was not a factor in choosing these reports. They were selected because both reports are typical of those produced from large-scale public inquiries and because they are both high profile accidents. Familiarity with the accidents will enable the reader to better judge the success of the language in modelling the behaviour.

1.1.6 Accident Report Structure

Accident reports vary in length, depending, amongst other things, on the scale of the accident and the depth of the investigation. The King's Cross report is nearly 250 pages long, with 21 chapters, 14 appendices, 31 photographic plates, and 14 figures. It contains summarised and detailed information on London Underground, its organisation, and events leading up to and during the accident. The chapters contain witness descriptions, expert opinions, and background information. The report concludes with a list of 157 recommendations, cross referenced to the chapter where they were first identified and annotated with a priority rating. However, incident reporting schemes can produce

reports as little as a few lines long. For example, the following text is the text of UK Air Accident Investigation Branch Bulletin No: 1/96 Ref: EW/G95/10/23 Category: 1.3:

> Towards the end of the landing roll, the aircraft developed a swing to the right which the pilot was unable to control. The aircraft ground looped bending the undercarriage legs and the propeller struck the ground. Weather conditions at the time were good and the pilot and passenger were not hurt in the accident.

This thesis focuses on the issues associated with large reports, although many of the problems addressed also exist in smaller reports.

1.1.7 Weaknesses of Accident Reports

Ensuring the quality of accident reports should be a high priority for organisations as they have a moral responsibility to prevent accident recurrence. They also have a financial responsibility to their investors; accident recurrence carries the possibility of damaging litigation and loss of customer confidence.

However, the structure, content, quality, and raison d'être of accident reports have been much criticised (e.g., [BJT97, LL98, BR92, LJ97a, TR94]). Charles Hoes, former president of the System Safety Society, comments:

> I have generally been unable to use many of the results of investigations because they either are unavailable, are not written in a way that I find useful for my work, or are wrong. (I am convinced that well over 75% of the investigations that I have reviewed are wrong in that they didn't accurately reflect the events and end up with incorrect 'causes' to the problem.) [Hoe96]

The following list provides an overview of aspects of natural language accident reports that have been identified as inhibiting the accurate communication of the report contents:

Size As discussed in Section 1.1.6, accident reports often contain large quantities of concise, detailed text. The sheer size of such documents makes it difficult for the reader to absorb all the salient points; a great deal of information can be forgotten, lost track of, or simply missed. The size also increases the chances of syntactic and semantic errors, ambiguities, and omissions in the transcription of the report.

Structure Sections of the report cannot be read in isolation. The common structural division of evidence and conclusions, as well as the use of rhetoric in reports [SJ98], require the reader to read the full document to ensure that all the information has been revealed.

Validation The presentation of current reports, as well as their structure and size, can obscure the validity of the report argumentation from both the author and the reader.

Natural language has no accepted formal syntax or semantics and so it is not currently possible for any mathematical analysis to be conducted over a natural language report.

Peer review and verbal descriptions are the most common means used by report writers for checking the consistency and coherence of the accident report content [BR91]. This

informal process does not ensure high quality in the argumentation; errors are still found in the validity of the reasoning in accident reports. For example, managerial wrongdoing is highlighted as a causal factor in the Challenger Shuttle crash, yet the report fails to address the question of why the management ignored the advice of their own engineers and made the decision to launch [Vau96].

Differing Viewpoints Every individual involved in the accident and investigation has a view on what happened in the accident and why. Mental representations of both witnesses and writers are affected by many complex contextual factors, such as knowledge and degree of involvement [Lep87b]. The evidence given by witnesses during the inquiry often conflicts with evidence given by other witnesses, who may have a different mental representation of the accident even though they were present during the same events. Reports are written by a number of authors, and the information presented in particular sections will be affected by the author's mental representation of the accident. Each view of the accident highlights different events that contributed to the accident; critical incidents in one chapter are often given lower priority or even omitted in other chapters.

The report is intended to be a representation of the 'inquiry's view', the consensus reached after presentation of all the evidence. However, this is not always possible. For example, on page 109, the King's Cross fire report states that 'the scientific evidence presented in Part Two of the Investigation still demonstrated a considerable divergence of views'. Whether explicit or implicit, inconsistencies can result in the reader building an inconsistent or incomplete mental representation of the accident.

Redundancy Repetition is common in reports: events may be summarised, and then described in more detail; the same events may be described from a different viewpoint in different chapters. Coherence and consistency therefore become an issue. Redundant detail increases the overall size of the report and there is no reliable means to check consistency between similar information.

As with some of the other weaknesses we highlight, redundancy is not always bad. In a large document, some repetition does help the reader.

Imprecision of Natural Language Sentences and terms in natural language may have more than one interpretation. The context of the sentence or common sense of the reader is often relied upon to extract the intended interpretation, but these are not always sufficient. If the semantics of a statement cannot be uniquely determined, the accuracy of the information communicated to the reader by the report cannot be assured. It is important that any two people (e.g., the report writer and a system designer) do not interpret an accident report differently. Some of the means by which this imprecision is introduced are as follows:

Ambiguity Structural and lexical ambiguity are both problematic in reports. Structural ambiguity arises because natural language phrases and sentences can have more than one valid parse tree. Lexical ambiguity arises from words with more than one meaning. Resolving the interpretation of terms, such as 'correct' and 'complete', may require knowledge of the report writer's geographical, cultural and vocational background. Lekberg [Lek97] presents further examples of how cultural and vocational differences can alter perceptions of accidents.

Where terms are defined in a report, the size of the report necessitates a glossary. The King's Cross fire report introduces definitions throughout the report, making them difficult to find. The Channel Tunnel fire report uses a glossary for some terms, and is far easier to use in this respect (although it fails to adhere to or use all these definitions in the body of the report).

Pronouns Pronouns, such as 'he', 'she', and 'it', rely on context for their interpretation.

Metaphor Metaphor, by its nature, introduces imprecision into the interpretation.

Synonyms Synonyms are extremely common and extremely problematic in accident reports. They are present in many forms:

Role/Name Confusion References to entities can vary between the referencing the role played and referencing the name of the entity. For example, in the Channel Tunnel fire report, HGV vehicle shuttle 7539 is also referred to as the incident train. This can confuse the reader and this confusion is exacerbated if the relationship between the roles and agents is never made clear. Further complications arise when an entity has many roles, particularly if these roles change over the course of the accident. For example, Pierre Desfray is referred to by name, as a member of the CTSA and as one of the Co-rapporteurs appointed to investigate the Channel Tunnel fire.

Natural Language Flexibility The dynamic nature of natural language makes the introduction and consistent use of abbreviated or alternative terms uncontrolled. In the Channel Tunnel fire report, the French Fire Commander is also referred to as 'the commander' and 'the incident commander'. The first seven chapters of the Channel Tunnel fire report refer to the Engineering Management System, the final two, which include all the recommendations, refer to the Equipment Management System. It is very likely from the context that these names have the same referent.

Temporal Referencing The form of temporal reference in accident reports can vary between direct ('it is 17.58'), indirect ('two minutes later'), and imprecise ('some time later') [JMW95]. In the Channel Tunnel fire report, references to 'all trains in the tunnel' implicitly require temporal information to determine the correct referent at that time.

Holonym Referencing A particular type of inconsistent referencing is holonym referencing. Rather than referring to a particular entity, the sentence refers to the entity of which it is a part. For example, the context of many of the references to the Rail Control Centre in the Channel Tunnel fire report strongly implies that the intended referent is actually one of its subsystems, such as the Engineering/Equipment Management System or the Rail Traffic Management System.

Passive Tense Passive tense sentences omit the subject from the sentence and it is not always possible to accurately infer the subject from the context. For example, in the Channel Tunnel fire report, the second paragraph of the Executive Summary states that:

> Concerning the safety of the Eurotunnel system overall, approval was given to engineering, equipment and rolling stock designs only after much scrutiny by appropriate specialists and the production of a detailed safety case.

No explicit indication is given of the approving body.

Abstract Actions Abstract terms, such as 'failed' and 'started', can hide the actual action that was performed, if the action is described in terms of its outcome.

Underdefined Values Words such as 'many' and 'short', as well as underdefined technical terms and wide ranging concepts, such as 'cognitive workload', have multiple interpretations [JMW95].

Representing Concurrency In a complex system, many different events can occur simultaneously. It can be important, during the analysis of the accident, to be aware of what is

happening in a particular time interval. Natural language is poorly suited to represent such concurrency. In a report, events are generally described in limited causal sequences, with independent concurrent events being described in separate paragraphs, or in separate chapters. Continual cross-referencing is required to follow the chronological order of events. As discussed in the paragraph on 'Differing Viewpoints', this can be made more difficult as, for a particular time frame, the traces of events described in different sections of the report may vary.

Identifying Key Agents and Actions An *agent* is 'something which has the capacity to change the environment' [Kho88]. Identifying the individuals or entities who contributed to, or responded to, a failure is a critical stage in any accident inquiry. Similarly, an explicit representation of critical actions helps to clarify the course of behaviour. The size, number of authors, level of detail, and scope of the report are all factors in making the identification of a consistent set of the agents and actions involved surprisingly difficult with natural language accounts [BJT97].

Distinguishing Prescriptive and Descriptive Behaviour The behaviour of the agents involved in an accident is unlikely to have been ideal or even expected. However, accident reports are not written to apportion blame, and this can make it difficult to ascertain from the text when certain behaviour is acceptable and when it is not. Commonly the best indication is the recommendations in the report, but as these are often located at the back of the report, they are little use in interpreting events as the report is being read.

Where the expected behaviour is described, the report often leaves implicit the information that the action actually occurred.

Incompleteness The scope of the accident can be defined as the chronological range between the first and last relevant event in the accident scenario. The separate viewpoints in each chapter can make the scope of the accident unclear. For example, in Chapter 12 of the King's Cross fire report, the chapter on the development of the fire, the first relevant event is viewed as being the ignition of the escalator. In Chapter 13, the chapter on safety management, the first relevant event is the failure of senior managers to consider recommendations from previous fires. The scope of the accident, as represented in Chapter 13, stretches back many years further than the scope represented in Chapter 12. If the reader misinterprets the scope, their visualisation of the accident will be incomplete and inaccurate.

Equally problematic is incompleteness *within* the scope. As stated previously, certain relevant information may be omitted in some chapters. The separation of this information within the report makes it difficult for the reader to construct a complete internal model of the relevant information.

Politics of Inquiries Although the stated aim of accident reports is to identify and analyse the causes of accidents to prevent their recurrence, many argue that a number of other factors affect the analysis. Rasmussen [Ras90] argues that analysis will stop when a 'familiar' cause is found. Lekberg [Lek97] and Maurino [Mau97] both state that the background of the analysts has a significant impact on the conclusions and recommendations of reports. Maurino further states that accident reports are produced 'to put losses behind, to reassert trust and faith in the system, to resume normal activities and to fulfil political purposes'. Witnesses in accident inquiries interviewed by Toft and Reynolds [TR94] agree that 'politically speaking you don't want inquiries to make their factual findings and come up with recommendations which cannot [be implemented], or give political difficulties in [implementation] and to a large extent inquiries are set up so that they come to the right conclusions'. Vaughan's investigation of the Challenger launch decision asserts that the

report blames 'success-blinded middle managers' because, to the public and the Presidential Commission, they are a more politically acceptable scapegoat than high ranking officials [Vau96]. In aviation accidents, Thomas [Tho92] reports that, statistically, the likelihood of pilot error being found as the cause is tripled where the pilot is killed in the accident.

Although it is not our aim to explicitly address these problems, in clarifying and correcting information and argumentation in the report, bias in the report becomes easier to identify.

Lack of Formal Procedure In the UK, the existing procedural framework for instigating public inquiries and reports has evolved over the last hundred years from an initial basis on legal investigations [TR94]. There are not always clear guidelines stating when a public accident investigation should take place. In the case of the King's Cross fire, the investigation was commissioned under the UK Regulations of Railways Act of 1871, which states that a formal investigation of the accident, and its causes and circumstances, will be held if it is 'expedient'.

Depending on the nature of the accident, a public investigation will be conducted under the provision of different legislative acts; an accident involving a train would be investigated under the Regulations of Railways act. An accident involving a commercial aeroplane would be investigated under the Civil Aviation (Investigation of Accidents) Regulations. Where there is no specific legislation, the Health and Safety at Work Act can instigate formal investigations. Inconsistency caused by the numerous separate mechanisms for instigating investigations can make the quality and content of reports extremely variable. In addition, the bureaucracy surrounding the production and distribution of reports can also cause problems. Drafts of public investigation reports are rarely distributed, even to the organisation under investigation [TR94].

In some industries, guidelines for conducting investigations and for formatting the report do exist, such as the ICAO report format [ICA93]. Even where the guidelines provided are tight, the lack of quality management in the investigation and report production processes results in unreliable and inconsistent output [BR91].

Collating and presenting the evidence of an accident into a concise and accurate textual form is a difficult process and the criticisms levelled at the process in this document are intended to be constructive in nature. Although we have identified a large number of weaknesses with accident reports, we demonstrate that the methodical construction of a formal model of the report prevents, improves, or makes explicit each of these identified weaknesses, with the exception of the lack of formal procedures. The features of the language and method that facilitate these improvements are discussed in Chapters 2 and 3. We do not argue that natural language is inappropriate for presenting accident reports, but that the accuracy and presentation of the information can be improved.

1.2 Formal Methods and Accident Analysis

The use of formal languages and methods is a common means to gain high confidence in the accuracy of information in the field of safety-critical system engineering. We propose using a formal method to model the behaviour of the system, as described in the report, for post hoc analysis. The models constructed in this thesis are intended to

play a similar role as existing models in accident reports, such as combustion models, that augment, rather than replace, the existing reporting procedures.

In this section, we define our terminology and outline the research into the use of formal languages and methods in accident report analysis. The EDAL language and SCS method, which are employed in this thesis, are then introduced, with a brief discussion of the motivation for choosing them for this application. Finally, there is a description of DALEX, an executable subset of EDAL.

1.2.1 Terminology

A *language* is a symbolic means of representing information. The symbols used may be graphically based (for example, data flow diagrams, fault trees, pie charts, and musical scores) or textually based (for example, numbers and computer programs). Different languages can express and emphasise different information from the same source. For example, data flow diagrams are used in conjunction with data dictionaries as the former expresses the movement of data, whilst the latter expresses the structure of the data.

A *formal language* is a set of symbols with formation rules that guide the structure of sentences and inference rules that define the manipulation of the symbols. Importantly, a formal language has a well-defined semantics. A *model* is an interpretation of a formal system. It is important to differentiate this 'model' from the outcome of the process of modelling an accident report. Where the interpretation is ambiguous, we will refer to the latter as a specification. A *specification* is an abstract description of the system as described in the report. In accident analysis, a *formal method* is a set of tools and formal languages used to unambiguously model the system, as described in a report. The method supports reasoning about properties of that model [HB95b]. In engineering terms, a method also has defined steps and heuristics for guiding model construction. In this thesis, we employ the latter type of structured method and thus benefit from additional guidance in constructing the model from the source. This is a contribution because previous approaches to accident report modelling lack much of this support.

1.2.2 Current Usage of Formal Methods

Since Floyd [Flo67] and Hoare's [Hoa69] proposals for proving program behaviour, the use of formal methods and languages in the design of computer systems has grown. They are increasingly used in system design, particularly safety-critical system design (see Clarke and Wing [CW96], and Hinchey and Bowen [HB95a] for a number of examples). For example, the Federal Aviation Administration's air traffic collision avoidance system (TCAS II) was specified in the formal language, Requirements State Machine Language, when it was discovered that a natural language specification could not cope with the complexity of the system [LHHR94].

The potential benefits of formal methods are accepted and their use is suggested or mandated by many industry standards [HB95b]. However, at present, the use of formal methods is largely restricted to initial system design, and their use later in the system development lifecycle is much less common [CBF+95]. Formal modelling of accident reports is one example of formal methods being used later in the lifecycle (see Figure 1.1).

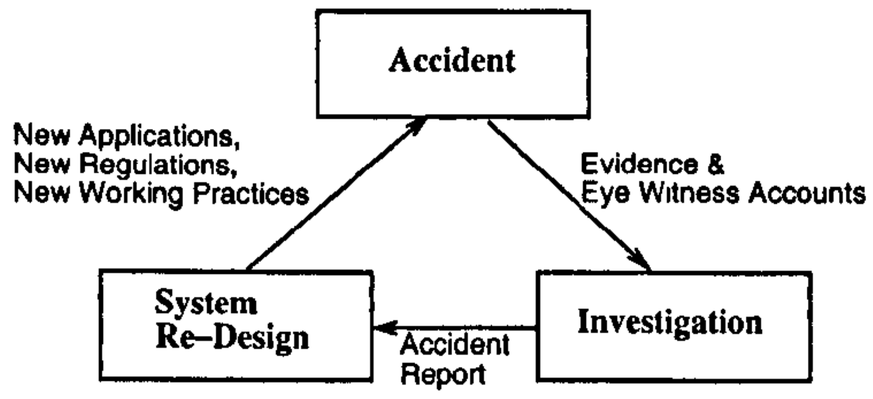

Figure 1.1: Design Lifecycle of a System Involved in an Accident.

1.2.3 Modelling Accident Reports

In the last few years a number of formal methods have been applied to modelling accident reports: first-order logic [Tho94], Petri nets [JMW95], CSP [FWH95], TLA [JT96], epistemic logic [Joh97a], the Conclusions-Analysis-Evidence notation [Joh97c], action logic [BJT97], Multilinear Events Sequencing [Ben97], fault trees [LJ97b], Object Z [BJ97], and WB-Graphs [LL98].

The different notations each proved to have advantages and disadvantages for representing different aspects of accident reports. For example, fault trees give an intuitive overview of the failures and errors in the accident, but are poor at representing event sequences. Action logics provide syntactic structures that highlight the behaviour performed, but are not easily understood by users unfamiliar with the logic. Formal notations have been used to help verify that the conclusions of a report follow from the body and that the body of the report is unambiguous, consistent, and complete [JT96].

Accidents, and the structure and content of the reports, vary so greatly that any one language is unlikely to be able to model everything that may be relevant to the accident. Indeed, such a language would not necessarily be desirable; by limiting the detail expressible in the model, the language focuses the attention on particular aspects of the report. Different languages may be more suited to modelling different accidents or types of accident.

General Formal Modelling Benefits

There are a number of general benefits that formal models of accident reports bring to the existing process. Each of the weaknesses of natural language highlighted earlier

was identified during the construction of a formal model of an accident report (described in Chapter 3). The identification of these problems can greatly increase the understanding of the system being modelled and is a strong argument for the use of formal languages for accident report modelling.

A formal language has a unique reading; a single parse tree exists for any sentence in the language. In translating the report into a formal language, inconsistencies, misunderstandings, and ambiguities in the information are revealed. These can be a result of the use of natural language or from informational incompleteness, such as described by Thomas and Ormsby [TO94], in their analysis of the interactions between side-stick controllers on the Airbus Industries A320:

> The activity of describing the design in LOTOS made us think very carefully, and hard, about the interaction between pilot control and the autopilot in more detail than was explicitly discussed in the requirements. For example, when the autopilot is disengaged, what is the control priority?

Formal languages represent information more concisely, by removing repetition, redundancy and reducing the necessary grammatical constructs. This process produces a more concise representation of the behaviour during the accident, in which it is easier to detect inconsistencies.

Formal models are necessarily more abstract than the natural language report; much of the report detail, such as historical information, is omitted from the formal model. This ensures a tighter focus upon the critical properties that directly led to an accident. The omission of unnecessary contextual detail produces a more general model of behaviour. In consequence, the model may be applicable to examining other scenarios than the accident scenario described in the report. Additionally, the results of analysing the model are more likely to be applicable to other systems. Identifying these 'critical properties' is a difficult task and must belong to accident investigators and domain experts and not to the analyst.

The model presented in this thesis was constructed without end-user involvement, and therefore confidence in the assumptions made about implicit, omitted, or conflicting information is reduced. This has also limited the evaluation that could be performed on the *modelling process*. This limitation is typical of speculative research activities, where end-user involvement is often dependent on an initial demonstration of the feasibility of the approach.

Scenarios of Use

The process of accident report modelling would ideally be performed by a team of analysts and domain experts working in conjunction, as recommended by Jones in [BBD+96]. As reported by Easterbrook et al. [ELC+98], the analysts in such teams expose implicit and inconsistent information, and reduce the risk of analysis bias in the model. The domain experts in the team are responsible for clarifying misunderstandings and identifying relevant behaviour.

This team approach will require significant resources. An alternative possiblity is that a single analyst is set the modelling task. With little or no domain knowledge, the analyst will have to make a number of assumptions, which must be documented. The case study presented in this document was constructed in a similar scenario to this. In Section 8.1.1, we discuss how these different categories of use could effect the modelling process.

There are a number of different approaches to accident report modelling. Johnson and Telford demonstrate that it can be used to examine consistency between different documents [JT96]. As we discuss in Section 7.1.3, modelling could also help the report writers manage the content of the report, prior to its publication. The work presented here is concerned with modelling the report content to enhance the visibility of specific information, to enable the behaviour to be animated, and to enable properties of the information to be proved. A model of the published accident report enables design teams to explore hypotheses about the system and accident. It can also indicate areas of the report that are incomplete or ambiguous. This will prompt the model builders to make assumptions or consult other information sources. Such a model could also be used to drive the production of addendums to an existing report and to support the criticisms that exist with the current accident report writing process.

Current State of Practice

It should be emphasised that we do not feel that the work presented in this document could be immediately adopted into current system design practice. At present, designers rarely use the accident report as the sole point of reference and there is evidence that many users of accident reports are sceptical of the findings [SJ99]. Such sceptisism is justified by a number of studies that have criticised the methods used within accident reporting bodies (for example, [BR85, LSS+99]). However, the work presented here is also useful for highlighting shortcomings in the report content. This helps the user to determine the information that should be sought from external sources.

In this document, we present an initial study of a particular modelling approach. Further work is required on developing the maturity of the method before it would be usable in the industrial domain (see Section 7.7.1).

1.2.4 Extended Deontic Action Logic: A Language for Modelling Accident Reports

This thesis explores the utility of using a deontic language for modelling and reasoning about accidents. Deontic logics were developed for reasoning about norms and their use was initially confined to reasoning about ethics and law [MW93]. However, they can be used more generally for reasoning about ideal and non-ideal behaviour, such as the behaviour commonly found in accidents. Accident reports generally describe norm-governed complex interactive systems in which some non-ideal, illegitimate, unexpected, or illegal behaviour has occurred: smoking in a 'no smoking' zone in the

King's Cross fire; arson in the Channel Tunnel fire. We describe such non-prescribed behaviour as *non-normative*. Unlike other logics, deontic logic allows the analyst to build models that 'report on bad behaviour rather than trying to banish it altogether' [Kho88]. We present detailed examples of this in Chapter 5.

We believe that the features of a deontic action-based logic make it a suitable language for modelling accident reports. When modelling an accident, it is important to represent both the expected and the actual behaviour of the system. By studying the disparity between these two, we can understand a great deal about why the accident occurred. Most formal languages cannot easily represent the 'contradictions' that often exist between the definitions of actual and expected behaviour. For example, given an action α, if we state in first-order logic that the action must not happen ($\neg\alpha$) and then that it does happen (α), we would end up with a contradiction. Of course, these languages can be extended for this purpose. Palanque and Bastide [PB97] demonstrate that forbidden behaviour can be expressed syntactically in standard Petri nets, using shading. Furthermore, Raskin et al. [RTvdT96] have extended the Petri net formalism with aspects of defeasible deontic logic.

Choosing Deontic Action Logic Given the number of deontic logics in existence, and the time and expertise required to develop one, the choice to evaluate existing deontic logics was straightforward. The language presented in this thesis, and referred to as Extended Deontic Action Logic (EDAL), is an extension of Deontic Action Logic (DAL[1]) [Kho88]. DAL is a requirements specification language framework developed as part of the FOREST (FOrmal REquirements Specification Technique) projects. These were major, internationally funded, industrially tested projects, with the aim of producing a formal specification technique for requirements engineering in real-time computer systems. In choosing DAL, we benefit from the associated methodological and tool support, and an associated body of published literature[2]. The definition of DAL in Khosla's Ph.D. thesis [Kho88] is comprehensive, and all references to DAL in this document are to the language as defined there.

The majority of deontic languages are closely linked to the algebraic standard system of deontic logic, KD, which in turn has close similarities to classical modal logic. These traditional deontic modal languages are riddled with paradoxes[3] (see [MW93] for a summary of Deontic Logic's history and problems). However, DAL takes the state-based approach of dynamic logic [Har84], rather than the traditional algebraic approach, and avoids most of these problems [Mey88].

There are a number of other deontic logics, such as the multi-agent logic ALX3 [HM95], which could be used as alternatives to DAL. In particular, the work of Fiadeiro and Maibaum [FM91] has strong links with DAL. Their work focuses on developing a state-based deontic logic that could be used for reasoning about safety and liveness

[1] In Khosla's Ph.D. thesis [Kho88], the logic is named DL, but later publications refer to DAL.

[2] The FOREST literature refers to Modal Action Logic (MAL), which extends DAL with temporal operators [Mai86].

[3] Deontic Logic is described by Struth as 'the most problematic and the least developed of the modal logics' [Str94].

properties. We chose to use DAL because it seemed less esoteric and we wish the formal models to be reviewable by non-logicians.

Introducing EDAL

- The underlying logic of EDAL is a many sorted first-order logic. The axioms for this logic are given in Appendix C.2.1.

- Benner and Rimson state that the 'building blocks' of an accident description should consist of pairs of actions and agents; what event took place, and who performed it [BR92]. The modal operator of EDAL explicitly defines these 'building blocks' in the syntax. The axioms for this modal operator, which include defining sequenced action descriptions, are given in Appendix C.2.2.

- The deontic operators of EDAL, based on the concepts of 'obligation' and 'permission' to perform actions, prescribe whether an action *should* or *may* be performed by an agent. Whereas most languages can model either the prescripted or the actual behaviour, EDAL can model both, and the relationship between the two. The semantics of the EDAL deontic operators differ from those of DAL, although the definitions are similar. These were changed partly to increase the flexibility and suitability of the language to accident report modelling, and partly to clarify the semantics. The axioms for the deontic operators are given in Appendix C.2.3.

The Strengths of EDAL In contrast to many languages, non-normative, or 'illegal', states are 'first class citizens' in deontic logic. An EDAL model can be used to consistently define and reason about actual behaviour both within and outside the prescribed behavioural constraints. This supports analysis of the reasons for, and outcome of, the divergence of the actual behaviour from the prescribed behaviour. In non-deontic languages, non-normative behaviour is either inconsistent or modelled as normal behaviour, thus insinuating that the behaviour is acceptable.

Actual behaviour need not differ from expected behaviour for it to be deemed a causal factor in an accident. In highly complex systems, the effects of the prescriptive behaviour are often unforeseen in certain modes of operation [Mel94]. An example from the King's Cross fire report is that after the report of a fire, the London Underground ticket office staff did not immediately contact the Fire Brigade. If this contravened the procedure then the training and response of the ticket office staff would be brought into question. However, London Underground did not have this policy and the response of the ticket office staff was consistent with official procedure. As this behaviour was deemed a causal factor in the accident, clearly the normative behaviour for the system was at fault. By highlighting normative behaviour that has been identified as a causal factor, the EDAL model emphasises the need for the prescripting entities, such as the management and the regulatory committees, to change the prescriptive behaviour of the system.

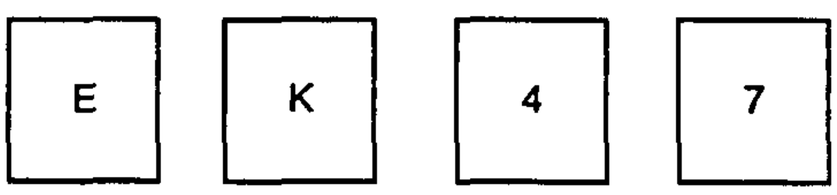

Figure 1.2: An Example of Wason's Selection Task

Non-normative actions do not necessarily result in an accident. EDAL is able to represent fault tolerance mechanisms, common to safety-critical systems. The effects of non-normative actions are not always immediately observable. EDAL is also able to consistently represent the occurrence of such *latent failures*. As latent failures and fault tolerance are commonplace in the systems described in accident reports, the ability to model them both consistently and reason about the resultant behaviour is a great benefit.

The deontic aspects of EDAL are not the only points of interest. We have mentioned the importance of agents in accident analysis. Agents are 'first class citizens' of EDAL, but not of most of other formal techniques used to model accident reports, such as Petri nets [JMW95], first-order logic [Tho94], and the Temporal Logic of Actions [JT96].

DAL was developed for industrial use and was required to be relatively easy to understand and to have methodological and tool support. Unfortunately, the constant development of the language meant that a full tool set was never developed. It is important that EDAL accident report models are reviewable by experts in the domain of the system being modelled, who are likely to be non-logicians. Clarke and Wing identify the accessibility of the model as fundamental to the success of its use [CW96]. The accessibility of EDAL models to non-logicians can be improved using techniques of literate specification and graphical support methods, as described in Section 3.3.2.

The presence of deontic conditionals in accident report models may make them easier to review. Work on Wason's four card selection task (see Figure 1.2) suggests that human reasoning with deontic conditionals is less error prone than with deductive conditionals (see [NE95] and [Cum96]).

1.2.5 Structured Common Sense

As well as demonstrating the benefits offered by formal accident report modelling, this thesis addresses practical concerns related to this approach. Many formal methods lack *scalability*, the ability to represent large, complex systems without losing structural clarity [BBD+96]. As both DAL and EDAL offer no support for structuring or encapsulation, both have poor scalability [LG97]. This is especially unfortunate as many accident reports are extremely large.

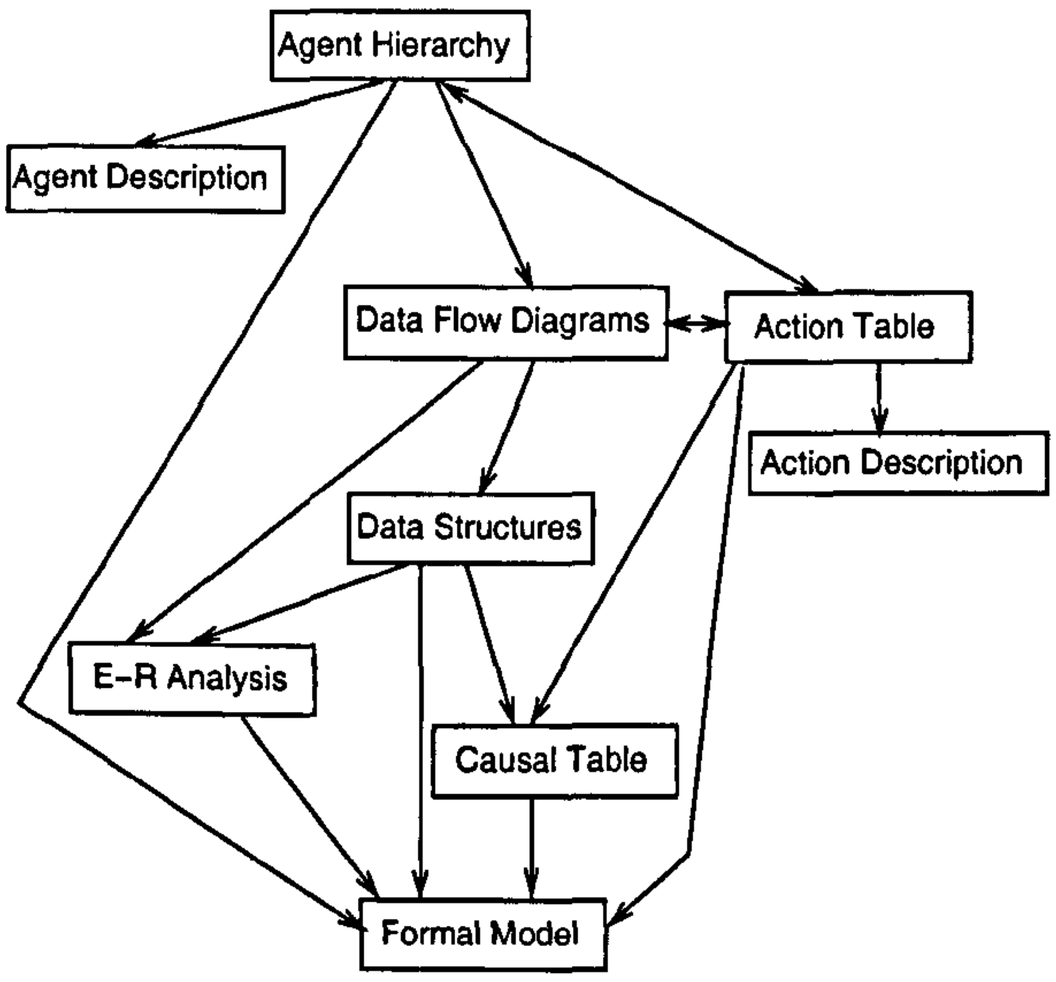

Figure 1.3: Dependencies between Steps of SCS

The 'Structured Common Sense' (SCS) [PFAB86] method is used in this thesis to construct models. It is an example of a structured method, which guides the activity of the analyst and thus ameliorates the problems of poor scalability. SCS was chosen because it was designed specifically to support MAL, an extension of DAL. The SCS method was designed for modelling the requirements of embedded real-time systems, and it needed some alteration for this novel application. Chapter 3 describes its use in guiding the construction of a large and complex model of the Channel Tunnel fire report. The method consists of eight steps, employing distinct modelling processes, such as agent hierarchies and action tables, to elicit or clarify particular aspects of the model, such as the relationships between the various agents and the actions performed by each agent. The stepwise approach focuses the analyst on a particular subset of the information in an accident report, helps to uncover particular types of errors, and requires particular types of modelling decisions. As well as aiding the construction of a consistent formal model, the intermediate representations improve the traceability of the system and make the information easier to review by non-logicans [ELC+98]. Figure 1.3 illustrates the order of the steps and the flow of information between steps.

1.2.6 Scoping the Model

We demonstrate that a more holistic modelling approach than that currently used in accident report modelling (such as, [JT96, LJ97a, Joh97c]) can significantly improve the model produced. We have described some of the categories of information in accident reports that are desirable to represent (agents, actions, time) and some of the languages that have been used to model these. There has been little research or discussion of

the issues involved in eliciting these categories from the accident reports and in constructing the formal models in a methodological way. This is surprising, given that a key motivating factor for this work is that this information is difficult to elicit from the reports.

The need to methodically construct accident report models has probably not been addressed because the models currently produced are intended to analyse the validity of a report conclusion against the evidence presented in the text of a report.

The advantage of this *conclusion validation* approach is that it typically produces small, low cost, formal models that are unaffected by the problems of scale. However, problems of scale do still affect the process of eliciting information from the accident report, and this approach relies heavily on the formalist manually searching through the report for arguments relating to the conclusion. Using only the conclusion validation approach, the analyst becomes familiar with a fraction of the report, making the already difficult task of identifying relevant information[4] even harder. The analyst may not uncover relevant implicit information.

Motivated by these problems, this thesis advocates a *wide scope* approach, in which the full report is modelled. The obvious disadvantages of this approach are the increased cost and the problems of scale. However, as demonstrated later, the combination of the SCS method with an interpreter for EDAL helps reduce the latter. The cost is not prohibitive either, as this approach does not require the full report to be *formally* modelled. Instead, only the initial semiformal modelling steps of the SCS method are performed over the entire report.

As demonstrated in Chapter 3, these early steps identify most of the information used in the formal model. More ambiguities, inconsistencies, and implicit and incomplete pieces of information are discovered due to the wider scope. The wider context assists in the resolution of inconsistencies and ambiguities. A further advantage of the wide scope approach is that the analyst gains a more accurate and complete understanding of the behaviour, whether it is explicitly described, omitted or implicit in the report [CW96]. The analyst focuses less on what information should and should not be included in the model than in the conclusion validation approach, which reduces the chance of confirmation bias affecting the model constructed [Ras90].

The scope of the model is necessarily reduced in the latter stages of the method, due to the poor scalability of EDAL. The EDAL formal model is therefore built to support analysis of specific hypotheses, as with the conclusion validation approach. However, because many of the informational flaws in the report, such as implicit or omitted information, have already been identified, made explicit, and resolved, we can have more confidence in the accuracy of the model. It is also possible to examine the validity of the both the explicit and implicit argumentation in the report. The analyst is also more qualified to generate other hypotheses for analysis, aside from checking the validity of the argumentation. Figure 1.4 illustrates the order and effect of these steps.

Unlike the conclusion validation approach, the initial modelling steps of the wide scope

[4] 'The hardest single part of building a ... system is deciding what to build', Brooks [Bro87].

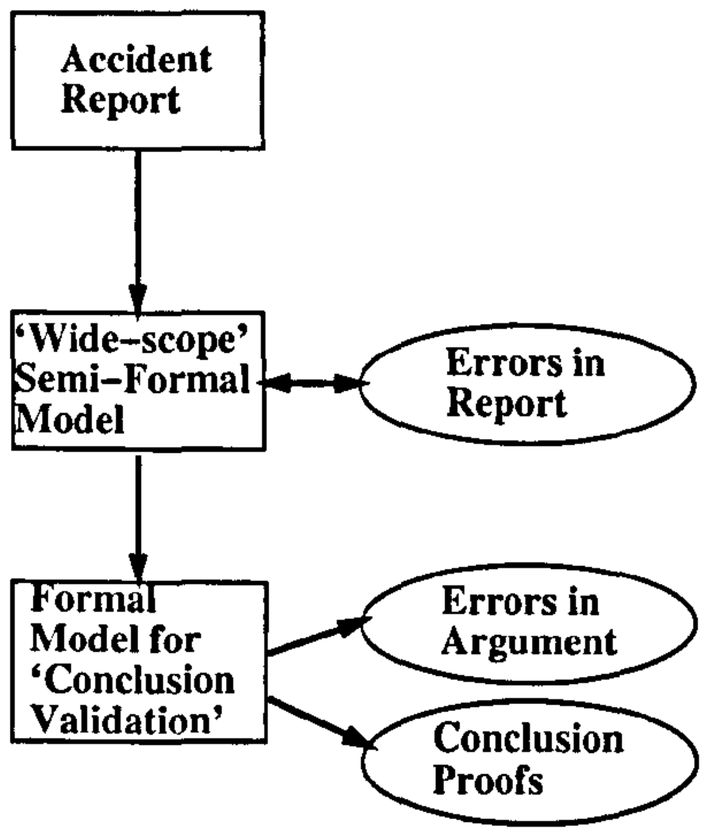

Figure 1.4: The Wide Scope Approach to Accident Report Modelling

approach are not focused on formalising a particular argument. Numerous formal models can be built from the output of the initial steps of the method. Furthermore, these formal models can be constructed in other languages than EDAL.

1.2.7 Executing EDAL

When developing EDAL, we wished to rapidly prototype the semantics of the language. Therefore, we constructed a Prolog-based interpreter for a subset of EDAL. This interpreter allowed us to experiment with different definitions of the operators, validate our intuitive understanding of their semantics. Ultimately, such an interpreter could also support formal reasoning in EDAL.

1.3 Contribution of Thesis

- We present what we believe is the first methodological approach to accident report modelling that encompasses elicitation and model construction. We discuss the benefits promised by the method. In particular, the incremental steps make the modelling process more straightforward and provide easy to understand representations of the formal model.
- We discuss the problems with the existing approach to accident report modelling and motivate a novel wide scope modelling approach. We demonstrate this new approach when we model the Channel Tunnel fire report. We highlight, in particular, the improvements to the accuracy of the formal model that this approach offers.

- We argue that similarities exist between the fields of requirements engineering and formal accident report modelling. We demonstrate that these can be exploited by applying a requirements engineering method to the construction of a formal accident report model. We add further traceability to the steps of the method to improve the link between the formal model and the report.

- By modelling the Channel Tunnel fire report, we emulate certain aspects of a realistic industrial application context. We discuss in detail the problems encountered while eliciting information and constructing formal models from this accident report. The large size of the report enables us to examine scalability issues in the modelling process.

- This work supports previous work criticising the accident reporting process, as the construction of the model highlights numerous ambiguities, inconsistencies, and errors in the report. We demonstrate that the activities of representing the information from the report in an unambiguous language and reasoning about this information help locate and highlight these informational flaws.

- We demonstrate that the deontic operators have an important role in accident report modelling, as they distinguish the prescriptive and descriptive information given in the Channel Tunnel fire report. In particular, they enable us to model the organisational safety frameworks and examine how they failed. We acknowledge the importance of identifying the actions and agents involved in an accident. We demonstrate that SCS and EDAL enable these to be identified and their roles in the model to be examined.

- We discuss the role of latent failures in accidents. We also describe the problems with the existing notion of normativity in deontic logics and present possible alternatives. We demonstrate within the model that the latent failure operators of EDAL enable us to differentiate between active and latent failures in the report.

- We present an interpreter that enables the operators of EDAL to be validated. We discuss its limitations and the requirements for other tools to support the method and language.

1.4 Structure of Thesis

In this chapter, we have described the weaknesses of the current reporting process and outlined the advantages offered by formal languages. EDAL is discussed with further technical detail in Chapter 2. In Chapter 3, EDAL is used to model the Channel Tunnel fire report. The steps of the 'Structured Common Sense' method, used to build the formal EDAL model, are each described and critiqued. Chapter 4 presents a formal model of the Channel Tunnel fire report. Chapter 5 presents example theorems and their proofs. Chapter 6 describes an interpreter developed for automating EDAL. The penultimate chapter of the thesis, Chapter 7, motivates further exploration and research

into a number of issues related to, but outside the scope of, the work presented here. Finally, Chapter 8 is a summary and evaluation of the work presented.

Part II

Deontic Action Framework

Chapter 2

Deontic Action Languages

EDAL is an extension of DAL [Kho88] for modelling accident reports. This chapter describes the DAL and EDAL languages. For convenience, the term (E)DAL is used when referring to both the DAL and EDAL languages. The description of EDAL in this thesis is self-contained. Although familiarity with the FOREST work is not necessary, a more detailed account of the DAL language than space permits here is presented in Khosla [Kho88]. DALEX is an executable language based on EDAL. The DALEX language is introduced, with a brief description of the aspects of EDAL that it includes. An interpreter for DALEX is described in Chapter 6.

2.1 Concepts

Before giving a technical description of the languages, we will outline the concepts underlying the languages.

Sorts

(E)DAL classifies all objects as belonging to a particular sort. For example, in the model of the Channel Tunnel fire, trains are modelled using the sort $Train$. Specific trains, such as train 7539 (the incident train), are modelled as constant values of sort $Train$, such as $train1$. The sort, or type, structure of (E)DAL constrains the domain of variables, constants, and predicates and the range and domain of functions.

There are a small selection of predefined sorts in (E)DAL, including Act and Agt. These represent the universe of action names and the universe of agent names respectively.

Scenarios

(E)DAL has an implicit (global) state object, which describes the currently derivable state information. DAL uses the term *scenario* to refer to the potentially incomplete information known about the state. Although this information is complete in EDAL, we will continue to use this terminology. The reasons for these differing approaches are discussed in Section 2.3.1.

Action Descriptions

An action description utilises a modal operator to describe a property known to hold following an action being performed by an agent (i.e., a post-condition). The operator, $[\_,\_]\_$, is of the form $[A,\alpha]\phi$, where A is an agent, α is an action and ϕ is a formula describing some property or properties that hold after A has performed α[1]. A conditional can be used to describe the scenario properties that must hold before the action description is valid.

Action Combinators

The action combinators of (E)DAL allow the structuring of sequenced actions, concurrent actions, and non-deterministic choice between actions. These make it possible to flexibly model the complex behaviour traces and interactions that exist in multi-agent systems, such as those described in accident reports.

Normativity

The deontic aspect of (E)DAL is concerned with describing and distinguishing between deontically acceptable (normative) and deontically unacceptable (non-normative) scenarios. Every scenario of every (E)DAL model is either normative or non-normative: a scenario is normative if the normative constant η holds in it; a scenario is non-normative if $\neg\eta$ holds. The (E)DAL deontic operators of obligation and permission define how the model moves between normative and non-normative scenarios.

The nature of normativity is dependent on the particular model, and the motivation behind the construction of the model. Normativity can represent morally, legally, or institutionally imposed constraints. The granularity of what is to be considered unacceptable is also flexible. For example, in the Channel Tunnel fire model, the striking Eurotunnel workers could be modelled as performing non-normative actions. Alternatively, if the model focuses on the behaviour of the Emergency Service personnel, the behaviour of these workers could either be omitted altogether or treated as normative. This decision is extremely important due to the limited binary nature of normativity. Later, we suggest ways to extend the notion of normativity to make it more flexible.

[1] Actions in (E)DAL are atomic and always terminate.

Permission

If an action is *permitted* to be performed by some agent, then it is known that if that action is performed by that agent in a normative scenario, the resulting scenario is normative. If an action is not permitted to be performed (*prohibited*) by some agent, then it is known that if that action is performed by that agent in a normative scenario, the resulting scenario is non-normative.

Permission and prohibition persist until explicitly revoked. However, as we discuss below, permissions and prohibitions are temporarily suspended in certain scenarios. Following Khosla [Kho88], we use the term *Permission structure*, or P-structure, to describe the set of permissions and prohibitions that hold in a particular scenario. In addition, we use the term *immediate permission* to refer to a scenario-specific permission to perform an action. An agent is immediately permitted to perform an action iff the agent is permitted to perform the action in the current scenario's P-structure. The term *weak permission* refers to the persistent permission of an action. Unless the agent is immediately obliged (see below) to perform an action, weak permission to perform an action implies immediate permission to perform that action. In DAL, weak and immediate permission are combined in a single permission operator.

Obligation

An obliged action is one that should be performed in this, or a future, scenario. In contrast to permission, an obligation to perform an action persists until it is revoked or until the obliged action has been performed. As with permission, we present two forms of obligation. If an agent is *weakly obliged* to perform an action, then the action should be performed by that agent in some future scenario, unless the obligation is revoked. If an agent is *immediately obliged* to perform an action, the action should be the next action performed by the agent. Weak obligation replaces immediate obligation in later versions of DAL, as described by Maibaum [Mai93].

The use of the term *should*, as opposed to *must*, reflects that an obligation is a deontic constraint rather than a behavioural one. When an agent is immediately obliged to perform an action, the P-structure that held prior to the immediate obligation is suspended. The new P-structure for that agent in that scenario contains only one (immediate) permission, namely to perform the immediately obliged action, and prohibitions from performing all other actions. If the next action performed by the agent is not the obliged one, then a non-normative scenario is reached. Following the performance of the obliged action (or its revocation by another agent), the suspended P-structure is restored, with any necessary alterations resulting from the performance of the obliged action and the behaviour of the other agents.

Seinsollen and Tunsollen Operators

The operators of (E)DAL are action based. Von Wright describe such operators as *Tunsollen*, as distinguishable from the more common *Seinsollen*, or state based operators of the traditional, modal logic based, deontic logics [vW80]. Khosla notes that the deontic operators seem intuitively suited to describing actions ('permission to do'), whereas other modal operators ('necessarily', 'eventually') seem intuitively suited to describing states. It is both possible and useful to define corresponding Seinsollen operators for (E)DAL, using the (E)DAL Tunsollen operators. This makes it possible to describe the permission to, obligation to, or prohibition from reaching certain scenarios.

2.2 Deontic Action Logic

The rules and axioms contained in this section are those given by Khosla [Kho88].

2.2.1 DAL Underdeterminedness

Underdeterminism is an important issue for modal logics. Pragmatically, a description of movement between states of a non-trivial system cannot exhaustively describe the effect or lack of effect it has on all the properties of the system. A common solution is the frame rule, which states that if a property is not explicitly affected, then it is preserved.

Rather than define a frame rule, DAL embraces this underdeterminism. An underdetermined action is an action with many potential implementations. In DAL, a scenario-transforming function models an action. An underdetermined action can therefore be represented by many different functions. DAL takes this approach because a DAL specification is intended to model the known information about a system, rather than choosing a particular implementation.

2.2.2 DAL Syntax

The deontic operator symbols for DAL are: O, P, $Pref$, OS, PS, Obl, Per, $Sobl$, $Sper$.

O Immediate Obligation

P Permission

$Pref$ Permission to refrain

OS Sequenced Immediate Obligation

PS Sequenced Permission

Obl Seinsollen Obligation

Per Seinsollen Permission

Sobl Seinsollen Sequenced Obligation

Sper Seinsollen Sequenced Permission

Formation Rules

In the following, $\mathcal{S}$ denotes the collection of DAL sorts, both predefined and user defined. Where the context is sufficient for an unambiguous interpretation, punctuation and subscripts have been omitted.

Terms :

- For each sort $s \in \mathcal{S}$, a variable or constant of sort s is a term.
- If $t_1 \ldots t_n$ are terms of sorts $s_1 \ldots s_n$ respectively and $s_1 \ldots s_n$ are all taken from $\mathcal{S}$ and f is a function symbol of sort $\langle s_1 \ldots s_n, s_{n+1} \rangle$ then $f(t_1 \ldots t_n)$ is a term of sort s_{n+1}.
- If $t_1 \ldots t_n$ are terms of sorts $s_1 \ldots s_n$ respectively and $s_1 \ldots s_n$ are all taken from $\mathcal{S}$ and a is an action symbol of sort $\langle s_1 \ldots s_n \rangle$ then $a(t_1 \ldots t_n)$ is a term of sort *Act*.
- Nothing else is a term.

Atoms (Atomic Formulae):

- If $t_1 \ldots t_n$ are terms of sorts $s_1 \ldots s_n$ respectively and $s_1 \ldots s_n$ are all taken from $\mathcal{S}$ and p is a predicate symbol of sort $\langle s_1 \ldots s_n \rangle$ then $p(t_1 \ldots t_n)$ is an atom.
- For each sort $s \in \mathcal{S}$, given two terms of s, t_1 and t_2, $t_1 =_s t_2$ is an atom, provided $=_s$ is an equality symbol of the language.
- Nothing else is an atom.

The Tunsollen deontic operators P, O, $Pref$, PS, and OS are predicate symbols.

Formulae :

- An atom is a formula.
- The logical constant, η, is a formula.
- If ϕ is a formula, so is $\neg\phi$.
- If ϕ and φ are formulae then $(\phi \vee \varphi)$, $(\phi \wedge \varphi)$, $(\phi \rightarrow \varphi)$, and $(\phi \leftrightarrow \varphi)$ are also formulae.

- If α is a term of sort Act, A is a term of sort Agt, and ϕ is a formula, then $[A, \alpha]\phi$ is also a formula.
- If σ is a term of sort $S(Act)$, A is a term of sort Agt, and ϕ is a formula, then $[\![A, \sigma]\!]\phi$ is also a formula.
- If ϕ is a formula and A is a term of sort Agt then $Per(A, \phi)$, $Obl(A, \phi)$, $Sper(A, \phi)$, $Sobl(A, \phi)$ are formulae.
- For each sort $s \in \mathcal{S}$, if x is a variable of sort s and ϕ is a formula, then $\forall_s x.\phi$ and $\exists_s x.\phi$ are formulae.
- Nothing else is a formula

FOL Axioms

With the exception of EQ, which admits non-rigid designators, the equality axioms, such as reflexivity, symmetry and transitivity, are omitted, but are as usual. If ϕ, ψ and τ are formulae, α, β are terms of sort Act, A is a term of sort Agt, σ is a term of sort $S(Act)$ then the following are axioms of DAL:

(1) $\phi \rightarrow (\psi \rightarrow \phi)$

(2) $(\tau \rightarrow (\phi \rightarrow \psi)) \rightarrow ((\tau \rightarrow \phi) \rightarrow (\tau \rightarrow \psi))$

(3) $(\neg\phi \rightarrow \neg\psi) \rightarrow ((\neg\phi \rightarrow \psi) \rightarrow \phi)$

(4) $\forall x \phi(x) \rightarrow \psi(t)$ where t is free for x in ϕ.

(5) $(\forall x(\phi \rightarrow \psi)) \rightarrow (\phi \rightarrow \forall x\psi)$
where x is not free in ϕ

(EQ) $\forall x, y((\phi(x) \wedge x = y) \rightarrow \phi(y))$
where y is not within the scope of a modal operator

Modal Operator Axioms

(6) $[A, \alpha](TRUE)$

(7) $([A, \alpha](\phi \rightarrow \psi)) \rightarrow (([A, \alpha]\phi) \rightarrow ([A, \alpha]\psi))$

(8) $([A, \alpha]\neg\phi) \rightarrow (\neg[A, \alpha]\phi)$

(9) $\forall x([A, \alpha]\phi) \leftrightarrow ([A, \alpha]\forall x\phi)$
where x is not free in A or α

(10) $\exists x([A, \alpha]\phi) \rightarrow ([A, \alpha]\exists x\phi)$
where x is not free in A or α

(11) $(([A, \alpha]\phi) \vee ([A, \alpha]\psi)) \leftrightarrow ([A, \alpha]\phi \vee \psi)$

(12) $(([A, \alpha]\phi) \wedge ([A, \alpha]\psi)) \rightarrow ([A, \alpha]\phi \wedge \psi)$

Basic Deontic Operator Axioms

(13) $\eta \rightarrow (P(A,\alpha) \leftrightarrow [A,\alpha]\eta)$
(14) $\neg[A,\alpha]\eta \rightarrow [A,\alpha]\neg\eta$
(15) $Pref(A,\alpha) \leftrightarrow \exists\beta(\neg(\beta = \alpha) \wedge P(A,\beta))$
(16) $O(A,\alpha) \rightarrow P(A,\alpha)$
(17) $O(A,\alpha) \rightarrow \neg Pref(A,\alpha)$

The characterisation of obligation given here reflects only its static aspects. The dynamic aspects of obligation are only expressible at the model level (see Section 2.2.4).

Action Combining Operator Axioms

(18) $[\![A,<\alpha>]\!]\phi \leftrightarrow [A,\alpha]\phi$
(19) $[\![A,<\sigma,\alpha>]\!]\phi \leftrightarrow [\![A,\sigma]\!][A,\alpha]\phi$
(20) $PS(A,<\alpha>) \leftrightarrow P(A,\alpha)$
(21) $PS(A,<\sigma,\alpha>) \leftrightarrow (PS(\sigma) \wedge [\![A,\sigma]\!]P(A,\alpha)$
(22) $OS(A,<\alpha>) \leftrightarrow O(A,\alpha)$
(23) $OS(A,<\sigma,\alpha>) \leftrightarrow (OS(A,\sigma) \wedge [\![A,\sigma]\!]O(A,\alpha)$
(24) $P(A,\alpha;\beta) \rightarrow ([A,\alpha;\beta]\phi \leftrightarrow [A,\alpha][A,\beta]\phi)$
(25) $P(A,\alpha \parallel \beta) \rightarrow (([A,\alpha]\phi \wedge [A,\beta]\psi) \rightarrow ([A,\alpha \parallel \beta]\phi \wedge \psi))$
(26) $\alpha \parallel \beta =_{Act} \beta \parallel \alpha$
(27) $P(A,\alpha+\beta) \rightarrow (([A,\alpha]\phi \wedge [A,\beta]\psi) \rightarrow ([A,\alpha+\beta]\phi \vee \psi))$
(28) $([A,\alpha]\phi \wedge [A,\beta]\psi \wedge \neg[A,\alpha]\psi \wedge \neg[A,\beta]\phi) \rightarrow (\neg[A,\alpha+\beta]\phi \wedge \psi)$
(29) $\alpha+\beta =_{Act} \beta+\alpha$

Seinsollen Deontic Operator Axioms

(30) $Per(A,\phi) \leftrightarrow \exists\alpha.[A,\alpha]\phi \wedge P(A,\alpha)$
(31) $Obl(A,\phi) \leftrightarrow \exists\alpha.[A,\alpha]\phi \wedge O(A,\alpha)$
(32) $Sper(A,\phi) \leftrightarrow \exists\sigma.[\![A,\sigma]\!]\phi \wedge PS(A,\sigma)$
(33) $Sobl(A,\phi) \leftrightarrow \exists\sigma.[\![A,\sigma]\!]\phi \wedge OS(A,\sigma)$

2.2.3 DAL Proof Theory

The meta symbol $\vdash$ denotes the provability relation. If Γ denotes a set of (DAL formulae) assumptions and ϕ denotes a DAL formula, then $\Gamma \vdash \phi$ states that ϕ is provable from Γ iff ϕ can be derived from the formulae of Γ and from the proof rules of DAL. A formula that is provable from an empty set of assumptions is a theorem.

The provability relation symbolising provability from a specific scenario is given as $\vdash_s$, where s is the theory presentation $< L, A >$ of the theory representing the scenario $t(s)$ (see Section 2.2.4).

The inference rules of DAL are:

Generalisation

$$\frac{\vdash_s \phi}{\vdash_s \forall x \phi} \tag{2.1}$$

Modus Ponens

$$\frac{\vdash_s \phi, \quad \vdash_s \phi \rightarrow \psi}{\vdash_s \psi} \tag{2.2}$$

2.2.4 DAL Model Theory

A DAL semantic model for a DAL *specification* is a set of scenarios and some scenario changing functions. In order to define what is meant by the terms 'semantic model' and 'specification', we will need to introduce the notion of theories and theory presentations.

Definition 1 (Theory Presentation) *A theory presentation TP is a pair $\langle L_{TP}, A_{TP} \rangle$. L_{TP} is a language description (i.e., the sorts and arities of operations) and A_{TP} is a set of axioms (in the language L_{TP}).*

DAL specifications and scenarios are theory presentations. For example, the specification *sp* is the following:

$$L_{sp} = \left\{ \begin{array}{lll} Sorts & = & Nat, Act, Agt, Book \\ Constants & = & \\ & & Tom : Agt \\ & & 0 : Nat \\ & & b1 : Book \\ Variables & = & \\ & & p : Agt \\ & & x : Nat \\ & & b : Book \\ Function\_Symbols & = & \_ + \_ : Nat \times Nat \rightarrow Nat \\ Action\_Names & = & \\ & & returnbook : Book \rightarrow Act \\ & & getbook : Book \rightarrow Act \end{array} \right.$$

$$A_{sp} = \left\{ \begin{array}{l} x + 0 = x \\ [p, getbook(b)]O(p, returnbook(b)) \end{array} \right.$$

In subsequent examples, we will omit L_{TP} (for a given TP) when it can be deduced from A_{TP}.

Definition 2 (Theory) *A theory TH of a theory presentation TP, denoted $TH(TP)$, is defined as the set of formulae provable from A_{TP}:*

$$TH(TP) = \phi : A_{TP} \vdash \phi$$

where ϕ is a formula expressed in the language L_{TP}.

$$TH(sp) = \begin{cases} x + 0 = x \\ 1 + 0 = 1 \\ 2 + 0 = 2 \\ 3 + 0 = 3 \\ 4 + 0 = 4 \\ 5 + 0 = 5 \\ \ldots \\ [p, getbook(b)]O(p, returnbook(b)) \\ [Tom, getbook(b1)]O(Tom, returnbook(b1)) \\ \ldots \end{cases}$$

Necessitation Rule

The Necessitation Rule is found in most modal logics. It states that if a property holds, with no extralogical assumptions, then it always holds (i.e., if ϕ is in the theory, $[A, \alpha]\phi$ is also in the theory).

The definition of this rule in DAL is complicated because both the specification and the potential scenarios are theories. The necessitation rule requires differentiating the provable properties of a specification theory from those of a scenario theory. This problem is solved by assuming that theorems are derivable from the following rule:

$$If\ \phi \in \Gamma\ then\ [A, \alpha]\phi \in \Gamma$$

where Γ is the minimum set of sentences that includes all the theorems of $TH(sp)$ and satisfies the above rule, the variables A and α range over the agent and action sorts, and ϕ is a DAL formula.

A theory that includes at least the same properties as another, is said to be an extension of the latter. In order to describe what a DAL model consists of, we must define both the concept of theory extension and the concept of scenario consistency.

Definition 3 (Theory Extension) *Given two scenarios s and s' (where $L_s = L_{s'}$), $TH(s')$ is said to be a theory extension of $TH(s)$ iff at least the same properties are derivable in s' as in s, written $s \subseteq s'$. Formally:*

$$\forall \phi.\ If\ \phi\ \in TH(s)\ then\ \phi \in TH(s')$$

Note, this implies that for a given specification *sp*, all theories are theory extensions of the scenario theory of the specification, i.e., $sp \subseteq s$.

Definition 4 (Consistency of Scenarios) *A scenario s is consistent iff it is not the case for any formula ϕ that both $\vdash_s \phi$ and $\vdash_s \neg\phi$ hold.*

A DAL model structure is the behavioural definition of the specification.

Definition 5 (Model Structure) *A model structure for a DAL specification sp is given by the pair $\langle \mathcal{D}, \mathcal{F} \rangle$ where $\mathcal{D}$ is a set of deontic scenarios and $\mathcal{F}$ is a collection of functions, one for each action name: $\mathcal{F}_{t(\alpha)} : t(\alpha) \times \mathcal{D} \rightarrow \mathcal{D}$ where $t(\alpha)$ is the sort of the action name.*

Before defining a DAL model, we must introduce some further model level operators, MOS and OOS. These are required to describe the dynamic aspects of obligation in DAL. There are two new sorts: AAP is a collection of agent-action pairs; $AS(Act)$ is a collection of sequences of agent-action pairs. In addition, there is a modal operator $[\![\_]\!]\_$, which takes the form $[\![\sigma]\!]\phi$ where σ is of sort $AS(Act)$ and ϕ is a DAL formula. Three new functions are also required: $< \_, \_ >$, $\langle\rangle$, and $::$. The axioms defining these operators are as follows:

(M1) $[\![\langle < A, \alpha > \rangle]\!]\phi \leftrightarrow [A, \alpha]\phi$
(M2) $[\![\sigma ::< a, \alpha >]\!]\phi \leftrightarrow [\![\sigma]\!][A, \alpha]\phi$
(M3) $OS(A, \langle < B, \beta > \rangle) \leftrightarrow (\exists\alpha.[B, \beta]O(A, \alpha)$
(M4) $OS(A, \sigma ::< B, \beta >) \leftrightarrow (OS(A, \sigma) \wedge \exists\alpha.([\![\sigma ::< B, \beta >]\!]O(A, \alpha)))$
(M5) $MOS(A, \sigma ::< B, \beta >) \leftrightarrow (OS(A, \sigma) \wedge \neg\exists\alpha.([\![\sigma ::< B, \beta >]\!]O(A, \alpha)))$

Definition 6 (Model) *Given a DAL model structure $\mathcal{M}$, this is a model for a DAL specification iff the following hold:*

(M6) $\forall s \in \mathcal{D}.\forall A \in Agt.\forall \alpha \in Act$
$if\ [A,\alpha]\phi \in TH(s)\ then\ \phi \in TH(f_{<A,\alpha>}(s))$

(M7) $\forall s \in \mathcal{D}.\forall A \in Agt.\forall \alpha \in Act$
$if\ \neg[A,\alpha]\phi \in TH(s)\ then\ \phi \notin TH(f_{<A,\alpha>}(s))$

(M8) $\forall s \in \mathcal{D}.\forall A,B \in Agt.\forall \alpha,\beta \in Act$
$if\ [A,\alpha]O(B,\beta) \notin TH(s)\ then\ O(B,\beta) \notin TH(f_{<A,\alpha>}(s))$

(M9) $\forall s \in \mathcal{D}.\forall A,B \in Agt.\forall \alpha,\beta \in Act$
$if\ A \neq B\ and\ O(B,\beta) \in TH(s)\ and\ [A,\alpha]\neg O(B,\beta) \notin TH(s)$
$then\ [A,\alpha]O(B,\beta) \in TH(s)$

(M10) $\forall s \in \mathcal{D}.\forall A,B \in Agt.\forall \alpha,\beta,\delta \in Act.\forall \sigma \in AS(Act)$
$if\ P(A,\alpha) \in TH(s)\ and\ O(A,\delta) \notin TH(s)$
$and\ MOS(A,\sigma) \in TH(f_{<B,\beta>}(s))$
$and\ \neg P(A,\alpha) \notin TH(f_\sigma f_{<B,\beta>}(s))$
$then\ P(A,\alpha) \in TH(s) \leftrightarrow P(A,\alpha) \in TH(f_\sigma f_{<B,\beta>}(s))$

(M11) $\forall s \in \mathcal{D}.\forall A,B \in Agt.\forall \alpha,\beta,\delta \in Act.\forall \sigma \in AS(Act)$
$if\ \neg P(A,\alpha) \in TH(s)\ and\ O(A,\delta) \notin TH(s)$
$and\ MOS(A,\sigma) \in TH(f_{<B,\beta>}(s))$
$and\ P(A,\alpha) \notin TH(f_\sigma f_{<B,\beta>}(s))$
$then\ \neg P(A,\alpha) \in TH(s) \leftrightarrow \neg P(A,\alpha) \in TH(f_\sigma f_{<B,\beta>}(s))$

Rules M6 and M7 describe the relationship between action descriptions and the scenario transforming functions. $f_{<A,\alpha>}(s)$ represents the scenario resulting from applying the function $f_{<A,\alpha>}$ to scenario s.

The rather complex rules of M10 and M11 reflect the dynamic aspects of DAL obligation. If an action is permitted or prohibited before a sequence of at least one obliged actions, then the action is still permitted or prohibited following the sequence, provided there have been no actions explicitly affecting it.

The nature of the obligation operator restricts DAL's approach to underdeterminism, as can be seen in semantic rule M8. Due to the way that obligations affect the P-structure, it must *always* be known if the following scenario contains an obligation. As described in M8, if it is not known that an obligation will hold following an action then the obligation does not hold following the action. M9 describes the persistence of obligation as other agents perform actions. These model rules (M6-M11) can, therefore, be viewed as very specific frame rules for DAL. Similarly, the normative status of the scenario must always be known, and axiom 14 of DAL ensures this.

2.3 Extended Deontic Action Logic

EDAL and DAL are very similar, sharing many axioms and formation rules. To reduce repetition and conserve space, we present only the sections of EDAL that differ from DAL; a complete description of the language is given in Appendices C and D. The numbers of the EDAL axioms are prefixed with the letter 'E'.

EDAL differs from DAL in two important ways. Firstly, EDAL provides additional syntactic operators. These simplify the semantics and make it easier to express certain concepts. Secondly, EDAL theories are assumed to be complete.

2.3.1 EDAL Determinedness

EDAL adopts a closed world assumption, in which theories are complete, because the system and behaviour described in accident reports are determined. However, the information given in the report is rarely adequate to model this complete system. While these information gaps are difficult to locate in a natural language report, the process of formalisation forces information to be made explicit and gaps are quickly found.

Identifying these gaps is a valuable process but, to study the behaviour of the system, they must be filled. With the help of domain experts, assumptions are made of expected values. The closed world assumption forces these assumptions to be explicit, leaving them open to further expert review. The behaviour of the system in alternative scenarios can be studied by assuming different values from those described in the report. For example, the behaviour of a system could be examined to see if the same events could occur once the recommendations of the report had been implemented. Another important benefit of this approach is that the reachability of scenarios can be determined. In DAL, this could not be calculated until an implementation was chosen.

The closed world approach is consistent across operators unlike DAL, which mixes underdeterminism with the requirement for complete information in the presence of obligation and normativity.

The closed world assumption ensures that the initial model is used, and this in turn provides EDAL with a set of induction rules. Another effect of this approach is that EDAL has a frame rule. A frame rule is required because action descriptions, in general, describe only a subset of the properties that hold in the subsequent scenario. This is a pragmatic approach to any non-trivial model of behaviour, but the described effects of an action can have implicit side effects affecting other unnamed properties. It is not desirable to have unconstrained information changes in the model and it is not practical to explicitly define the value of all properties after each action. Most state based formalisms have some 'frame axiom/rule' that formally or informally adds a minimal change constraint (i.e., Ξ schemas in Z). The concept of minimal change is problematic as there may be many distinct minimal changes and, in practice, this requires extensive validation by domain experts.

EDAL's frame rule states that if a property ϕ is known to hold in the current scenario, ϕ also holds in the scenario following, provided the effect of the action performed does not contradict ϕ (see Clause 2.4 in Section 2.3.3).

2.3.2 EDAL Syntax

EDAL Formation Rules

The formation rules of EDAL are largely unchanged from DAL, the main difference being that the predefined agent sort Agt is omitted in EDAL. Instead, EDAL has a meta-sort of agents, in which many user-defined agent sorts can be defined (e.g., passengers, staff). This permits more localised definitions of agent sorts; staff and passenger agents can be defined with distinct prescriptive and descriptive behaviour. The term Agt is used in EDAL to refer to all defined agent sorts; for example, the modal operator is still of the form $[A, \alpha]\phi$ where A is of the meta-sort Agt, α is of sort Act and ϕ is an EDAL formula.

The deontic operator symbols for EDAL are: IO, IP, O, P, OS, PS, Obl, Per, $Sobl$, $Sper$. IO and IP are predicate symbols of EDAL.

IO Immediate Obligation

IP Immediate Permission

O Weak Obligation

P Weak Permission

OS Sequenced Immediate Obligation

PS Sequenced Weak Permission

Obl Seinsollen Immediate Obligation

Per Seinsollen Immediate Permission

$Sobl$ Seinsollen Sequenced Obligation

$Sper$ Seinsollen Sequenced Permission

EDAL Modal Operator Axioms

The FOL axioms of DAL are unchanged in EDAL. However, there are changes to the modal operator axioms. The assumption of complete information makes the implication in axioms E7, E8, E10, E11 and E12 valid in both directions:

(E6) $[A,\alpha](TRUE)$
(E7) $([A,\alpha](\phi \rightarrow \psi)) \leftrightarrow (([A,\alpha]\phi) \rightarrow ([A,\alpha]\psi))$
(E8) $([A,\alpha]\neg\phi) \leftrightarrow (\neg[A,\alpha]\phi)$
(E9) $\forall x([A,\alpha]\phi) \leftrightarrow ([A,\alpha]\forall x\phi)$
where x is not free in A or α
(E10) $\exists x([A,\alpha]\phi) \leftrightarrow ([A,\alpha]\exists x\phi)$
where x is not free in A or α
(E11) $(([A,\alpha]\phi) \vee ([A,\alpha]\psi)) \leftrightarrow ([A,\alpha]\phi \vee \psi)$
(E12) $(([A,\alpha]\phi) \wedge ([A,\alpha]\psi)) \leftrightarrow ([A,\alpha]\phi \wedge \psi)$

EDAL Basic Deontic Operator Axioms

The permission operators of EDAL separate the two concepts of immediate and weak permission. This has the major advantage of enabling immediate obligation to be fully characterised at the syntactic level.

The two obligation operators are used to express immediate and weak obligation. We note that FOREST industrial trials indicated that the constraint on the immediacy of the action is too strong for the formalisation of some requirements [Mai93]; obligation was weakened in later FOREST logics. The presence of both forms of obligation in EDAL increases the flexibility of the language, making it easier to express certain concepts. However, it may also confuse readers who are unfamiliar with EDAL.

The deontic operator definitions for obligation and permission of EDAL are as follows:

(E13) $\eta \wedge IP(A,\alpha) \rightarrow [A,\alpha]\eta$
(E14) $\neg[A,\alpha]\eta \rightarrow [A,\alpha]\neg\eta$
(E15) $IO(A,\alpha) \rightarrow IP(A,\alpha)$
(E16) $IO(A,\alpha) \wedge \neg[A,\alpha]IO(A,\alpha) \rightarrow [A,\alpha]\neg IO(A,\alpha)$
(E17) $P(A,\alpha) \wedge (IO(A,\beta) \rightarrow (\alpha = \beta)) \rightarrow IP(A,\alpha)$
(E18) $\neg P(A,\alpha) \wedge \neg IO(A,\alpha) \rightarrow \neg IP(A,\alpha)$
(E19) $O(A,\alpha) \rightarrow \exists\sigma[\![A,\sigma]\!]\neg O(A,\alpha) \wedge PS(A,\sigma)$
(E20) $O(A,\alpha) \wedge \neg[A,\alpha]O(A,\alpha) \rightarrow [A,\alpha]\neg O(A,\alpha)$

E13 has been weakened slightly from DAL, to allow the inclusion of the latent failure operators (see below). E14, and E15 are much the same as in DAL, except that IP and IO replace P and O respectively. However, this changes the definition of immediate obligation quite significantly. Immediate obligations affect only the scenario-specific immediate permissions, and not the persistent weak permissions. The performance of an obliged action directly alters the permissions or prohibitions in the system. These changes manifest themselves immediately, even if the agent is still obliged, and can affect other agents.

E16 and E20 describe the revocation of obligation (immediate and weak respectively) following the performance of the obliged action. This must be explicit, due to EDAL's frame rule. For the same reason, there is no requirement for an axiom defining the persistence of weak permission.

E17 defines the relationship between weak and immediate permission: a weakly permitted action is immediately permitted unless some other action is immediately obliged in that scenario.

E19 defines weak obligation: if an action is weakly obliged, there exists a permitted sequence of actions following which the action is no longer weakly obliged.

Latent Failures and Conditions

Reason discusses the differences between active failures that 'have a direct impact on the safety of the system' and latent conditions that may not manifest themselves until long after the action was performed[2] [Rea97]. Latent conditions are resident in all complex systems and are causal factors in complex system accidents.

A *latent condition* is the effect of some non-normative action that does not affect the behaviour of the system until later. A *latent failure* is the non-normative action that leads to a latent condition. Poor staff training was a latent condition in both the Channel Tunnel fire and the King's Cross fire. Note that a latent condition can be an *error of omission* (i.e., the effect of an agent failing to perform the expected action) or an *error of commission* (i.e., the effect of an agent performing an action they were not permitted to perform).

Latent conditions cannot be represented in DAL. However, EDAL provides an extended notion of normativity in which it is possible to differentiate the underlying normativity of the system (*absolute* normativity) and the *observable* normativity of the scenario.

By absolute normativity, we refer to the simplistic binary notion of normativity used by DAL and other deontic logics. With absolute normativity, the performance of any prohibited action or non-performance of an obliged action will lead to a non-normative scenario. We found that reasoning about absolute normativity in the systems described in accident reports was rarely worthwhile, as latent failures quickly make the scenario non-normative.

By observable normativity, we refer to a notion of normativity in which the performance of a prohibited action or non-performance of an obliged action does not necessarily lead to a non-normative scenario. For example, if the obliged management agent fails to perform the immediately obliged action $train(staff)$, and this action is defined as a latent failure, the scenario remains observably normative. Figure 2.1 uses Reason's 'Swiss cheese' metaphor [Rea97] to demonstrate that loss does not necessarily result from latent failure alone.

Although this approach has limitations, we found it useful to differentiate latent failures from active failures and to be able to reason about the observably normative behaviour up until the system's defences are breached. This approach enables the analyst to model

[2]Reason compares latent conditions to resident pathogens in the human body. They may lie dormant for years before combining with active failures and a particular set of circumstances to penetrate the system's defences.

the differing effects of latent and active failures and to examine how they interact to produce a loss event.

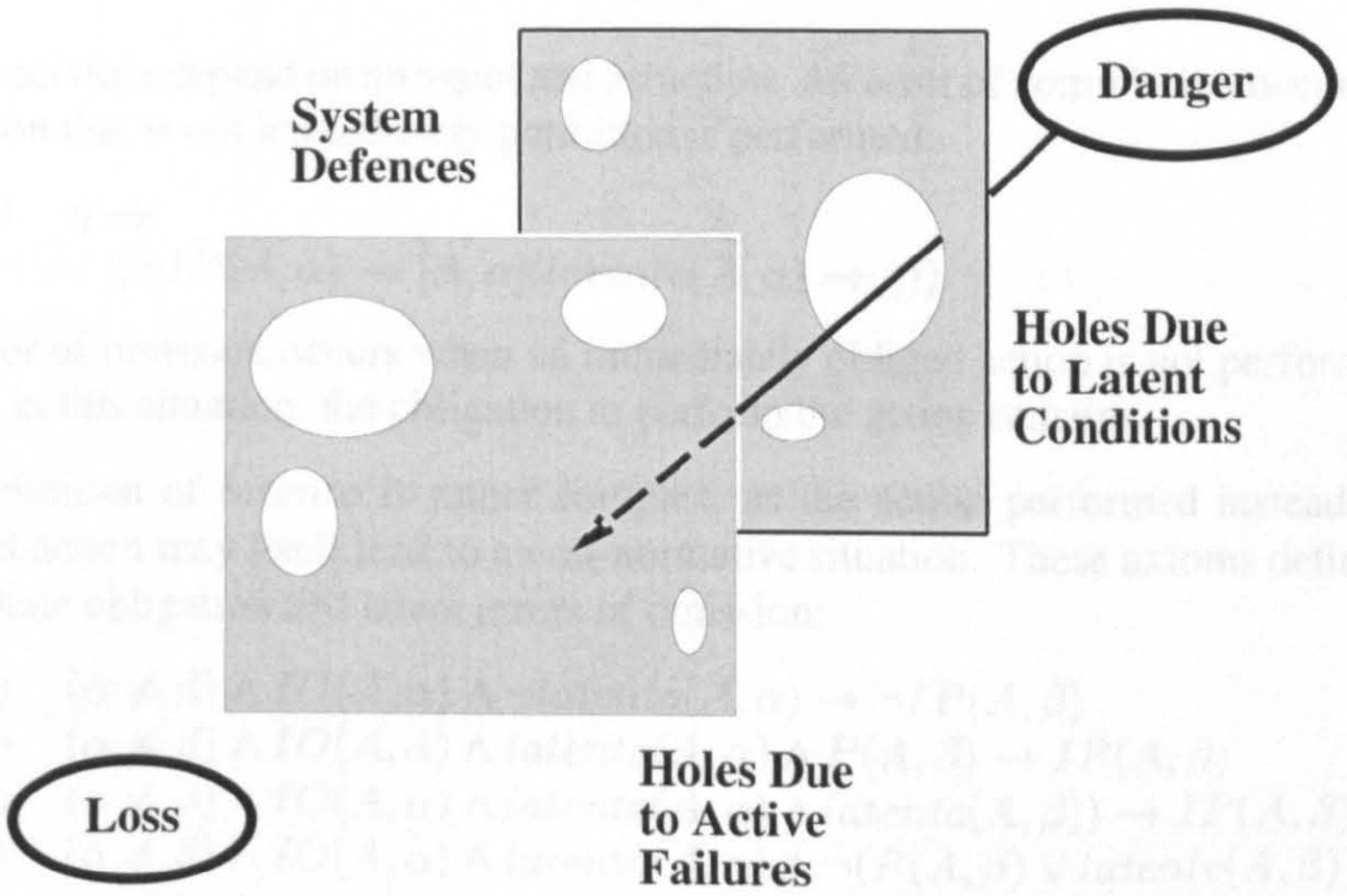

Figure 2.1: Reason's Swiss Cheese Model Representing Latent Conditions Without Loss

Using EDAL's latent failure operators, the current scenario reveals the observable normativity of the system and the occurrence of latent failures reveal hazards that underlie this observable behaviour. A non-normative EDAL scenario is reached through a non-normative non-latent failure or the triggering of a latent condition in conjunction with some other events. The analyst can also examine the absolute normativity of the system in isolation, by omitting latent failure definitions from the model.

The latent failure operators provide one means to avoid the problems of Contrary-to-Duty imperatives [Chi63]. As an example, the Eurotunnel management are obliged to ensure the safety of the passengers and staff in the event of a fire. In addition, the Eurotunnel Management are obliged to ensure there are no fires. If there is a fire, a conflict occurs. Von Wright states that:

> Such contradictions cannot be 'solved' in logic, only in the practice of norm-giving. [vW83]

The latent failure operators alter the practice of 'norm-giving' to reflect observable normativity. Defeasible deontic logics, which dynamically adjust the norms used to ensure consistency, are more commonly used to avoid these problems. Unfortunately, these logics are poorly suited to accident analysis, as described in Section 5.4.2.

The latent failure operators are:

latentc Latent failure leading to error of commission

latento Latent failure leading to error of omission

Both operators depend on an agent and an action. An error of commission occurs when an action that is not immediately permitted is performed.

(E21) $\eta \rightarrow$
$(\neg IP(A,\alpha) \rightarrow [A,\alpha](latentc(A,\alpha) \rightarrow \eta))$

An error of omission occurs when an immediately obliged action is not performed. In EDAL in this situation, the obligation to perform the action remains.

The definition of *latento* is rather complex, as the action performed instead of the omitted action may itself lead to a non-normative situation. These axioms define both immediate obligation and latent errors of omission:

(E22) $(\alpha \neq \beta) \wedge IO(A,\alpha) \wedge \neg latento(A,\alpha) \rightarrow \neg IP(A,\beta)$
(E23) $(\alpha \neq \beta) \wedge IO(A,\alpha) \wedge latento(A,\alpha) \wedge P(A,\beta) \rightarrow IP(A,\beta)$
(E24) $(\alpha \neq \beta) \wedge IO(A,\alpha) \wedge latento(A,\alpha) \wedge latentc(A,\beta)) \rightarrow IP(A,\beta)$
(E25) $(\alpha \neq \beta) \wedge IO(A,\alpha) \wedge latento(A,\alpha) \wedge \neg(P(A,\beta) \vee latentc(A,\beta)) \rightarrow \neg IP(A,\beta)$

The Committee on the Safety of Nuclear Installations created a classification of error types [RPM+81]. This states that, as well as errors of commission and omission, observable errors can also result from tasks being performed at the wrong time and tasks being performed inaccurately [RPM+81]. EDAL is not a temporal logic and the focus of the thesis is not on temporal reasoning, so we omit an operator for latent conditions resulting from temporal issues. We refer to failures resulting from inaccurate performance of a task as qualitative failures. The definition of a generic latent failure operator for qualitative failures proved difficult, as actions and properties have no fixed arity and the qualitative failure could occur in any property of the action. As demonstrated in Chapter 4.2, qualitative failures resulting in latent conditions can be modelled as commission and omission failures.

Action Combination and Seinsollen Operator Axioms

Apart from the numbering (E26 to E41), only the reference to IO instead of O separates the definition of the combination and Seinsollen operators in DAL and EDAL. DAL axioms 13 and 14 are also included in this section, as they define how actions are combined in the modal connective.

2.3.3 EDAL Proof Theory

The closed world assumption affects the proof theory of EDAL. To characterise it properly, we need to introduce the non-provability operator, $\underline{\vdash}$. The DAL proof theory is extended with the following rules:

$$\frac{\vdash \phi}{\underline{\vdash} \neg\phi} \tag{2.3}$$

$$\frac{\vdash \phi,\ \underline{\vdash}[A,\alpha]\neg\phi}{\vdash [A,\alpha]\phi} \tag{2.4}$$

If an EDAL formula is known to hold, its negation cannot be proved also to hold (Clause 2.3). Using the rule of double negation, this formula also states that if the negation of an EDAL formula holds, the formula cannot be shown to hold. If an EDAL formula ϕ holds in the current scenario, and it cannot be shown that following an agent A performing an action α the negation of ϕ holds, then ϕ holds following the performance of α by A (Clause 2.4).

2.3.4 EDAL Model Theory

EDAL model theory is similar to DAL model theory. One important difference is that EDAL explicitly differentiates Specification Theories and Scenario Theories:

Definition 7 (Specification Theory) *Given an EDAL specification sp, $SpTH(sp)$ is inductively defined by:*

1 $(A_{sp} \vdash \phi) \rightarrow \phi \in SpTH(sp)$

2 $(\phi \in SpTH(sp) \wedge A : Agt \in L_{sp} \wedge \alpha : Act \in L_{sp}) \rightarrow [A,\alpha]\phi \in SpTH(sp)$

Note that the definition of the specification theory includes the necessitation rule for specifications. In DAL, this is only assumed to hold.

For example, for a specification sp with the language description L_{sp} and the axioms A_{sp}, the specification theory, $SpTH(sp)$, is as follows:

$$L_{sp} = \begin{cases} Sorts & = & Nat, Act, Agt, Book \\ Constants & = & \\ & & Tom : Agt \\ & & 0 : Nat \\ & & b1, b2 : Book \\ Variables & = & \\ & & p, q : Agt \\ & & x : Nat \\ & & b : Book \\ Function\_Symbols & = & \_ + \_ : Nat \times Nat \rightarrow Nat \\ Action\_Names & = & \\ & & returnbook : Book \rightarrow Act \\ & & getbook : Book \rightarrow Act \end{cases}$$

$$A_{sp} = \begin{cases} x + 0 = x \\ [p, getbook(b)]O(p, returnbook(b)) \end{cases}$$

$$SpTH(sp) = \begin{cases} x + 0 = x \\ 1 + 0 = 1 \\ 2 + 0 = 2 \\ 3 + 0 = 3 \\ 4 + 0 = 4 \\ 5 + 0 = 5 \\ \ldots \\ [p, getbook(b)]O(p, returnbook(b)) \\ [p, getbook(b)][q, getbook(b)]O(q, returnbook(b)) \\ [Tom, getbook(b1)]x + 0 = x \\ \ldots \end{cases}$$

Definition 8 (Scenario Theory) *Given a specification sp and a scenario s, where s and sp share the same language, $TH_{sp}(s)$ is the set of the formulae that are provable using the scenario axioms (i.e., A_s), the FOL and EDAL axioms, and the formulae in the specification theory. Formally:*

$$TH_{sp}(s) = \{\phi \mid A_s \cup SpTH(sp) \vdash \phi\}$$

Given the specification sp from the previous examples, a scenario s and its scenario theory $TH_{sp}(s)$ could be given as follows:

$$s = \begin{cases} O(Tom, returnbook(b1)) \\ O(Tom, returnbook(b2)) \\ O(Tom, returnbook(b1)) \rightarrow P(Tom, returnbook(b1)) \end{cases}$$

$$TH_{sp}(s) = SpTH(sp) \cup \begin{cases} O(Tom, returnbook(b1)) \\ O(Tom, returnbook(b2)) \\ O(Tom, returnbook(b1)) \rightarrow P(Tom, returnbook(b1)) \\ P(Tom, returnbook(b1)) \end{cases}$$

The axioms of the specification are necessary truths that hold in all possible worlds. The scenario theory contains both these and the scenario specific truths.

In subsequent examples we will omit sp from $TH_{sp}(TP)$ as the scenario theories we discuss always use the same specification theory. We will also use the term *theory* as shorthand for scenario theory. Note that the theory of a specification, $TH(sp)$, is equivalent to the specification theory of a specification, $SpTH(sp)$.

EDAL Model structures are defined similarly to DAL model structures (see definition 5). However, the definition of an EDAL model is much more concise than that of a

DAL model, partly because EDAL has no requirement for model level rules defining the dynamic aspects of obligation. In DAL, the obligation operator cannot be fully axiomatised at the syntactic level, as it explicitly removes, suspends, and restores the P-structure. Instead, DAL constrains obligation fully at the model theory level. EDAL's two permission operators remove the requirement to alter the persistent permissions when altering the P-structure for an obligation. This enables the full axiomatisation of the immediate obligation operator at the syntactic level, and consequently enables reasoning about immediate obligations at the syntactic level.

Definition 9 (EDAL Model) *An EDAL model structure $\mathcal{M}$ is a model for an EDAL specification sp iff the following trans-scenario property holds in $\mathcal{M}$:*

> *If the theory $TH(s)$ contains an action description stating that the property ϕ holds after agent A performs action α, then ϕ holds in the theory accessed from $TH(s)$ by the performance of α by A.*

$$\forall s \in S, \forall A \in Agt. \forall \alpha \in Act.$$
$$if\ [A,\alpha]\phi \in TH(s)\ then\ \phi \in TH(f_{<A,\alpha>}(s))$$

2.4 Executable Extended Deontic Action Logic

Executable Extended Deontic Action Logic (DALEX) is an executable subset of EDAL. The version of DALEX presented here incorporates the modal operator and the main deontic operators of EDAL: immediate permission; immediate obligation; weak permission; weak obligation. DALEX also incorporates EDAL's assumption of complete information.

A full discussion of the code of the DALEX interpreter is presented in Chapter 6. However, we will quickly outline how the DALEX language compares to DAL and EDAL. The operators are as follows:

`ip(X)` Immediate permission

`io(X)` Immediate obligation

`p(X)` Weak permission

`o(X)` Weak obligation

`modal(X,Y)` The action description operator, which takes an action X and a DALEX formula Y.

Note that the operators do not explicitly link actions with agents.

EDAL theories are modelled as a list of DALEX formulae. A property `X` holds in a theory `DB` if the property is a member of the list: `member(X, DB)`. When a new scenario is entered, procedures are called to determine what holds in the new scenario. As in EDAL, properties persist by default, except for immediate permissions. To ensure the list is finite, sorts must not be inductively defined.

DALEX models execute a fixed trace of actions. If the formula `modal(X,Y)` holds in the current scenario and `X` is the next action to be performed, then `Y` holds in the following scenario. Negation as failure is used to model the closed world assumption: if `Y`, in the action description `modal(X,Y)`, is a negative property, for example `neg(p(stop(train)))`, `p(stop(train))` is removed from the scenario following action `X`. Note that `neg` differs from the Prolog `not` operator, and is used in DALEX to explicitly negate properties. If a formula `io(X)` holds in a scenario, then `ip(X)` and `modal(X,neg(io(X)))` also hold. If there are no immediate obligations in the current scenario and `p(X)` holds, `ip(X)` also holds. The action combination, Seinsollen and latent failure operators are not currently defined in DALEX.

2.5 Summary

This chapter has introduced the three deontic action-based languages referred to in this thesis and described how they relate to each other. DAL was introduced in detail. The description of EDAL built upon the description of DAL and described the differences between the two languages. In the following chapter, we discuss the application of EDAL and give added justification for the choice of language and the extensions. The description of the DALEX language is given here to demonstrate its limited nature in comparison to EDAL. In Chapter 6, we present a Prolog based interpreter for DALEX.

The content of this chapter should be understood by analysts wishing to use EDAL. Accident investigators working in conjunction with the analysts should have an informal understanding of the semantics of the operators.

Chapter 3

Methodical Construction of a Case Study

3.1 Introduction

Methods that structure and guide activity are used in engineering in the development of systems of a non-trivial nature. However, this structure and guidance is often missing in formal methods [HB95b]. The construction of formal models of the complex behaviour described in most accident reports is a difficult task. Surprisingly, there has been little mention of structured methodological modelling techniques in the literature discussing the application of formal methods to accident analysis. We introduce and evaluate the 'Structured Common Sense'(SCS) method, applying it to the construction of a formal model of the Channel Tunnel fire report [All97].

Given the high profile of this accident and the report, ambiguities and errors should be less common than in other reports, making it is easier to focus on the utility of EDAL and SCS. By electing to model real reports, rather than a narrative summary or a report fragment, the utility of SCS and EDAL are tested in a more realistic application context.

All quotes from the Channel Tunnel fire report are given with a reference to the paragraph and chapter, the paragraph number first. The labels 'Intro', 'Exec', and 'CTSA' refer to the sections entitled 'Introduction', 'Executive Summary', and 'Channel Tunnel Safety Authority', which precede the numbered chapters of the report.

In the initial steps of the method we use the wide scope approach to modelling, as described in Section 1.2.6. In the later steps, we reduce the scope and use a hypothesis driven approach, similar to the conclusion validation approach.

3.2 Requirements Engineering and Accident Analysis

SCS [PFAB86] is a requirements engineering method for eliciting requirements and modelling them in MAL. Using a requirements engineering method to assist with accident report modelling makes it possible to capitalise on the similarities between the tasks. Accident report modelling aims to produce a complete, consistent, unambiguous model based on the behavioural description given in the report. Requirements engineering is the discipline of attempting to develop a complete, consistent, unambiguous specification of requirements, based on information elicited from a variety of sources.

Eliciting information from the report is comparable to eliciting requirements from clients. In the former, information is elicited from different sections of the report, in the latter, from different clients. In both, this information is generally at varying levels of detail, contains inconsistent terminology, and can be inconsistent with other information. It is vital to minimise errors in the elicitation, specification and validation of the requirements, as correcting these errors later in the development lifecycle incurs a high cost in terms of time, capital, and system complexity [CK92][1]. Accident report modelling has similar concerns; we wish to identify and, if possible, resolve errors in the report. We wish to avoid introducing errors in the model construction process. The more accurate the model of the report, the more reliable the results of the analysis. Finally, both activities elicit a large quantity of information that must be stored and managed sensibly.

There are also differences between the requirements engineering and accident report modelling. One advantageous difference is that, in accident report modelling, the information does not need to be prompted from the report and it does not alter over time. However, because a report is passive, it cannot be prompted further if the information elicited is incomplete, ambiguous or inconsistent. Although it is extremely important to identify such informational flaws, we also wish to model and reason about the report. This requires using domain knowledge and common sense to clarify or make assumptions about these informational flaws. As we discuss throughout this chapter, building a model of the report without the involvement of a domain expert was very difficult. However, evaluating the construction of a model with end-user involvement is another thesis in itself. The difficulties we encountered without a domain expert are also likely to be faced by other readers unfamiliar with the system, such as interested parties from other industries.

3.3 Features of 'Structured Common Sense'

SCS is a design method, similar to CORE (COntrolled Requirements Expression) [Mul82], JSD (Jackson System Development) [Jac83], SSADM (Structured Systems

[1] Indeed, Mellor reports that incidents and accidents are commonly the result of poor requirements detection and validation [Mel94].

Analysis and Design Method) [LN87], and OOSE (Object-Oriented Software Engineering) [JCJÖ92]. As with these other methods, SCS encourages the structured development of the specification through a framework of sequentially ordered steps. The order of these steps are displayed in Figure 1.3. In modelling the Channel Tunnel fire report, we found that the agent hierarchy and the action tabulation steps elicited the majority of the information used in the model.

SCS is designed to guide the analyst through the development of the model, each step focusing on a particular aspect of the system. However, there are a number of other properties that make the method advantageous for accident report modelling.

3.3.1 Learnability and Ease of Use

There is significant expense involved in introducing formal methods to an existing process [HB95b]. Methods that require less training reduce this cost. Rather than develop SCS from scratch, the techniques were selectively 'cannibalised' from common requirements engineering methods, such as CORE and JSD [PFAB86]. User familiarity with these steps ensures reduced requirements for skills transfer. The use of common techniques also increases the likelihood of there being existing tool support.

3.3.2 Readability

A specification of a complex system should be both 'clear and realistic' and 'formal and rigorous' [Har88]. Using EDAL facilitates the latter, but it does not produce specifications as easy to understand by non-logicians as, for example, Petri nets or state chart specifications. Such readability is particularly important to this application, as the models must be reviewable by domain experts, who may be unfamiliar with modal logic. The overall readability of the model is improved by the use of well-known techniques, such as data flow diagrams and action tables, in conjunction with techniques of literate specification, such as natural language descriptions of the agents and actions in the system, to complement the formal language based step.

3.3.3 Scalability

The size and complexity of reports, and the systems they describe, demand methods and techniques to handle issues of scale in the construction and use of the model. The poor scalability of many formal methods is related to poor support for guiding and organising the activity of the analyst through the specification process [PFAB86]. Such methods typically describe features of the outcome of the process, without advising on methodical means to reach that outcome. The steps of SCS are each designed to focus the processes of elicitation and model construction on a particular type of information in the report. The multiple, narrowly focussed, steps enable the incremental construction of models of the size and complexity found in accident reports.

used here is as described in the FOREST report published by Potts et al. [PFAB86]. This version of the method is chosen because the report presents each step in detail. As the steps use common, well-documented techniques, this chapter will only briefly introduce each step. We focus on how the nature of the information in the report affects the application of each step.

To illustrate the various issues addressed in a large scale modelling process, this section describes the steps taken in the construction of the Channel Tunnel fire report model. Unfortunately, space limitations prevent the inclusion of the output of the full modelling process, which is over 200 pages in length. Examples are therefore drawn from a subsection of the behaviour described in the report. For the reader's benefit, this report subsection features high level entities and the behaviour does not require in-depth knowledge of the Channel Tunnel system.

3.4.1 Eliciting Agents

In both accident report writing and modelling, it is important to identify all agents who contributed to or responded to the accident, such as the individuals responsible for maintaining and operating safety systems [BR92, BJT97]. SCS's initial steps focus on eliciting and representing the agents of the system. As well as identifying the agents in the report, these steps highlight sections of the report in which it is not possible to incontrovertibly identify the referenced agents. The techniques used are the agent hierarchy (based on that of CORE) and natural language descriptions.

The Agent Hierarchy

The hierarchy is a structured representation of the agents and the organisational structures that group them. The initial iteration of the hierarchy was developed by reading through the report, recording all entities involved. Non-agent entities are removed from the hierarchy once the later steps of the method link actions with agents.

Further iterations of the hierarchy develop the relationships between the agents, starting at the highest level, and gradually build up a tree structure. In CORE, the agent hierarchy is based on the organisational or functional relations between the agents. The Channel Tunnel fire hierarchy is based on the former, as the system best fits the criterion of involving 'direct provision of services to numerous people, perhaps in several different organisations' [Mul82]. The branch nodes of an organisational hierarchy are departments and teams, the leaf nodes are people and equipment. This provides the reader with an accessible overview of the agents in the system and their organisational structure. The contextual information provided by neighbouring nodes can be useful for interpreting the role of each agent in the model. Figure 3.1 demonstrates the hierarchy created from the agents in the Channel Tunnel fire report introduction (agents in bold). The first of the five paragraphs from the introduction is given here as an example of the source material:

3.3.4 Traceability

A drawback of incremental development using multiple steps is that inconsistency can be introduced between steps. In the field of requirements engineering, there is a recognised need to trace requirements between steps but there are a number of associated problems [GF93]. One problem is that there is no agreed definition of what requirements traceability is. We use Gotel and Finkelstein's [GF93] orienting definition:

> Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases).

SCS does not explicitly support traceability, but the design of SCS does emphasise incremental formalisation. The methodological steps build up the formal model gradually: first eliciting the objects of the model, then the relations and actions, and finally the behaviour. The steps are ordered so that, if possible, each step feeds information to support the one following. This incremental development facilitates tracing information through the steps of the method. Finkelstein and Potts [FP86] assert that automated support for traceability would be straightforward to implement in SCS.

The ability to trace the information in the formal model back to more easily understandable representations of the model can assist the user in visualising the model. Gurr [Gur95] demonstrates that graphical methods linked to a formal specification can assist in clarifying assumptions and correcting misinterpretations, and, in some cases, can support automating reasoning.

3.3.5 Matching the Information Needs of EDAL and Accident Analysis

The FOREST projects were concerned with developing techniques for specifying multi-component computer based systems. SCS is designed to assist with constructing these types of models in MAL. Each step is intended to elicit information directly related to the syntactic categories or structures of a MAL specification.

It is clearly advantageous to use a method that is designed for a language so similar to EDAL. However, as is shown below, there are differences between the intended and current application that affect the information we wish to elicit and how we elicit it. Some of the steps of SCS are altered slightly from their original usage to suit the new application domain.

3.4 SCS Steps: Constructing the Model

Details of the SCS method are available in a number of other publications and the method differs slightly between these ([PFAB86, FP86, FP88]). The version of SCS

> At 21.58 hrs on 18 November 1996, Heavy Goods Vehicle shuttle No 7539, the incident train, travelling from France to the United Kingdom, stopped in Running Tunnel South, approximately 19 km from the French portal, with a fire on board. The fire emitted large quantities of smoke which rapidly engulfed the Amenity Coach at the front of the train. There were 31 passengers and two crew members in the Amenity Coach, and the train driver was in the front locomotive. The smoke enveloped the train drivers cabin and the Amenity Coach. The passengers and crew members all suffered from smoke inhalation and the presence of smoke made their evacuation to the safety of the Service Tunnel extremely difficult. (1/Intro)

- Eurotunnel
  - Eurotunnel Staff
    - Drivers
      - **Incident Train Driver**
      - **Evacuation Train Driver**
    - Chef de Train
      - **Incident Train Chef de Train**
    - Steward
      - **Incident Train Steward**
    - STTS Vehicle Drivers
      - **STTS Ambulance Driver**
      - **STTS Vehicle Driver**
    - Emergency Response Teams
      - **French First Line of Response**
- Emergency Services
  - **Medical Team**
  - **Transfer Drivers**

Figure 3.1: Agent Hierarchy Elicited from the Report Introduction

Agent Descriptions

Natural language descriptions provide further relevant detail about each identified agent. This easily referenced information is a valuable means to clarify the role of the agents featured in the report and formal model. In the report, information on agents is spread out and can be difficult to locate. As described in Section 1.1.7, inconsistent referencing between the agent name and role can lead to confusion.

The length of descriptions vary, depending on the complexity of the role of the agent, the assumptions made about the behaviour of the agent, and on other information that adds relevant context to the agent. Where possible, the descriptions directly quote the text of the report to minimise chances of misinterpretation.

Figure 3.2 provides an example extract from the Channel Tunnel fire report agent descriptions.

- **CTSA** The Channel Tunnel Safety Authority comprises national delegations of safety experts from the two countries. It advises and assists the Intergovernmental Commission on all matters concerning safety of the Fixed Link. It handles the examination, investigation, and reporting of safety incidents in the tunnel. More details of the individual members are given in the report.
- **Co-Rapporteurs** The co-rapporteurs were assigned by the CTSA to investigate the incident in question. They are directly concerned with the handling the inquiry and producing the report. The co-rapporteurs were also members of the CTSA, but we consider this a separate role.

Figure 3.2: Extracts from the Agent Descriptions

There were some difficulties with performing these steps with the Channel Tunnel fire report, mainly as a result of the fixed and limited information available in the document. However, as we discussed earlier, we do wish to identify the information that is omitted from the report.

Agent Interpretation Difficulties

Building an agent hierarchy focuses the analyst on the relationships between the various named entities in the report, the actual entities referred to by the names, and the higher level groupings of these entities.

Many of the problems in performing this analysis on an accident report are reflected by problems in identifying agents during requirements elicitation (i.e., ambiguity, synonyms, pronouns). These problems are highlighted earlier, during the discussion of the weaknesses of accident reports in Chapter 1.1.7, in the section on the 'Imprecision of Natural Language'. Further problems are caused by the information in the report requiring domain expertise. The bi-national nature of the Channel Tunnel system (and the report writers) introduces further problems with inconsistent referencing. For example, the French Incident Control Centre is referred to as the 'Poste de Commandement Avancé' in some of the chapters of the report.

In a number of cases in the Channel Tunnel fire report, agents are referred to, but their organisational affiliation is not given. The Immigration Police are one example. No indication is given whether these are Eurotunnel employees, emergency service personnel, members of the French Government, or belong to some other institution. The behaviour of the Immigration Police described in the report is quite varied and provides little assistance. This issue was eventually clarified after an examination of the French translation of the Channel Tunnel fire report, in which corresponding sentences refer to the Immigration Police as the DDCILEC. Although this term is not used in the body of the English translation of the report, the glossary of the English version defines the DDCILEC as the French Police. The Immigration Police are thus classified as emergency service personnel in the hierarchy.

By identifying synonyms and missing information, the hierarchy highlights potential

weaknesses in the report and enables a clearer and more complete picture of the organisational relationships to be constructed.

Resolving Difficulties

Many of the difficulties initially encountered in identifying agents can be resolved using information elsewhere in the report. For example, in a number of cases, actions are described passively without any agent being explicitly mentioned. In most of these situations, the immediate context enabled an assumption to be made regarding the identity of the agent. However, in the following example, from Paragraph 1, Chapter 1, the context provides no further information:

> The loading of the lorries onto the Heavy Goods Vehicle shuttle took about 20 minutes from 21.20 to 21.40 hrs.

A new 'Unidentified Loading Agent' was created in the agent hierarchy. In the Channel Tunnel fire model, most of these unidentified agents were eventually attributed to identified agents; in this case the staff working in the terminal are enumerated in the report in Paragraph 1, Chapter 2.

Scalability

The size of the Channel Tunnel fire report created problems for the elicitation steps. It was difficult to keep track of what had been elicited, and this made it harder to check for synonyms or conflicts. For example, the initial agent hierarchy was extremely large, and thus difficult to construct, check, and analyse; Data Flow Diagrams (used in the next step of SCS) with numerous nodes can result in a large, complex, tangled and difficult to read diagram. We decided that the best solution was to divide the report into smaller sections and perform the elicitation steps of SCS on these sections.

After the elicitation steps were completed for the report sections, the output from the sections was merged to represent the information in the full report. The small agent hierarchies produced in these more manageable sections of the report are merged to form a global agent hierarchy. Where there were potential synonyms, such as with Engineering Management System and the Equipment Management System, the contexts were known and could be easily checked. For reference, the full hierarchy from the Channel Tunnel fire is presented in Appendix F. Only the Data Flow Diagrams are not merged.

Representing the Hierarchy

The diagrammatic technique advocated by SCS and CORE (see Figure 3.3) for representing the full agent hierarchy was unsuitable to this application. The large number of agents in the report made the graphical hierarchy extremely large. This made it

difficult for the reader to visualise the structure of the system and to navigate the hierarchy. We adopted a more concise, structured textual representation, using indentation to represent the levels of the hierarchy (as demonstrated by Figure 3.1).

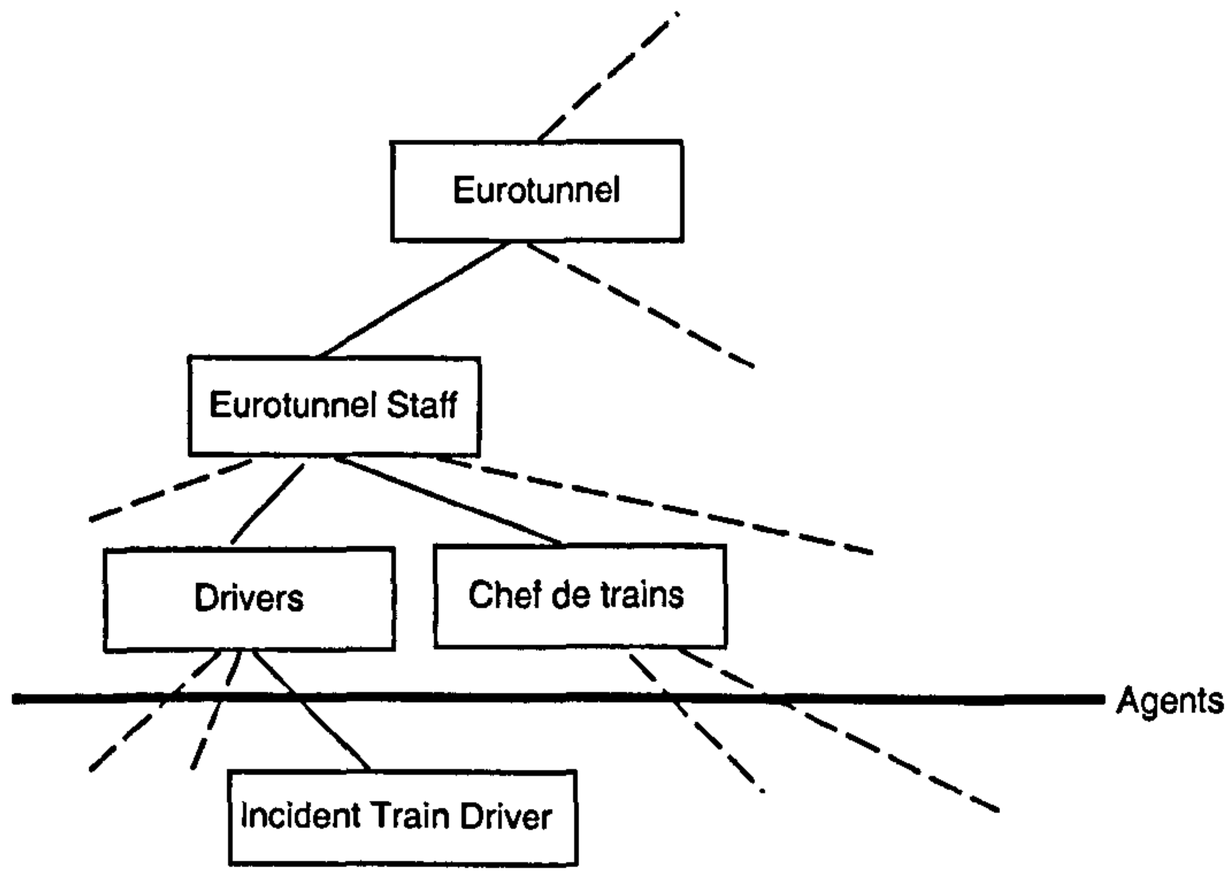

Figure 3.3: Small Example of SCS Diagrammatic Hierarchy

Capturing Source Information

To improve the traceability of the information in the agent hierarchy, we extend each agent entry with a reference to its location in the report. This makes it much easier for the reader to locate references to specific agents in the report because the agents are much easier to locate in the hierarchy.

The identification of the source in the report should facilitate the validation of assumptions and context checking, by both the analyst and domain experts. In many cases during the agent elicitation, decisions had to be made between possible interpretations of the report. For example, we assume that the French Fire Commander agent is also the referent of other terms such as 'the Fire Commander' and 'the Incident Commander'. Where a decision about an ambiguous referent is made, it is highlighted in the hierarchy (see Figure 3.4). This is vital, as subsequent reasoning is dependent on these assumptions.

French Fire Commander 31/3, 33/3(implicit), 38-41/3, 44/3('the French Incident Commander', 'the Incident Commander') 46/3, 51/3(the Fire Commander); 6/8('French Commander'), 17/8(implicit); 23/9(implicit)

Figure 3.4: References to the French Fire Commander in Accident Report

This feature also permits concordance-like studies on the distribution of mentions for

each agent. Analysis of such distributions can be used to give a rough indication of the importance of the role played by the agent in the accident. For example, from the entry for the First Line of Response from France, given in Figure 3.5, it is possible to determine that the agent played an important part in the accident (from the number of references), particularly in the middle to latter stages (from the profusion of references to the middle and later paragraphs in Chapter 3), and that some aspect of their behaviour is likely to be amended (from the reference to a report recommendation). An overview of the focus of each section of the report can also be generated. However, these are extremely crude measures, as neither takes the context of the reference into account.

French First Line of Response 2/Introduction; 1/2; 4/1('Emergency Services'), 20/1, 29/1; 23/3, 26/3, 27/3, 28/3, 29/3, 30/3, 34/3(the French Team), 21/3(assume from 22/3 context), (15, 34, 40, 41, 46, 47, 50, 52, 55 /3(implicit); 2/4; 18/5; 2/8, 3/8, 14/8, 15/8, 18/8(implicit), 42/8('the First Line of Response team'); 22/9; Recommendation 16.

Figure 3.5: References to the French First Line of Response in the Accident Report

Agents as Leaf Nodes

With EDAL's flat sort structure, only the leaf nodes of the hierarchy are modelled as agents. This can create problems as branch nodes in one section of the report are referred to as agents in other sections. For example, the UK First Line of Response is generally described in the report as a single entity, although details are given, in Chapter 3 of the report, of reconnaissance teams constructed from its members. It would be unwieldy to simultaneously refer to many agents (i.e., all the personnel in the First Line of Response) as performing a single action, as each agent would need to be referred to in an explicit conjunction.

In some cases, a branch node is used as shorthand for referring to one or more leaf nodes. For example, the Rail Control Centre, which is represented as a branch node, contains both the Engineering Management System and the Rail Traffic Management System, which are represented as leaf nodes. The Engineering Management System and Rail Traffic Management System control specific sections of the system: the Engineering Management System controls lights, cross-passage doors and other fixed equipment; the Rail Traffic Management System monitors and controls signalling, speed limits, and train separation. However, the report consistently refers to the Rail Control Centre as performing these actions. As it would (probably) be inaccurate to attribute these actions to the Rail Control Centre, the assumption is made in the model that such actions are actually performed by particular systems within the Rail Control Centre.

In some cases, the leaf nodes do not represent a complete breakdown of the members of the branch node. For example, references to the Rail Control Centre also describe actions that the Engineering Management System and Rail Traffic Management System would not perform (for example, radio communication with the Chef de Train of the incident train). In these situations, we found that the behaviour ascribed to the

'branch node agents' (i.e., the Rail Control Centre) is not related to that performed by the individual 'leaf node agents' (i.e., the Rail Traffic Management System and the Engineering Management System). Distinct leaf nodes are introduced to represent the 'branch node agent' performing these other actions (see Figure 3.6).

Eurotunnel Systems/Centres
...
UK Terminal
...
Rail Control Centre
Engineering Management System
Rail Traffic Management System
Rail Control Centre

Figure 3.6: Representing 'Branch Node Agents' in the Hierarchy

3.4.2 Eliciting Actions

The actions described in an accident report are as important to model as the agents. The actions describe the behaviour that led to the accident. SCS employs three methods to elicit and define actions: data flow diagrams, action tables, and action descriptions.

Constructing the data flow diagrams and action tables requires an examination of the relationships between the actions in the system and the agents who initiate, perform, or are affected by them. Once again, it is important to relate the names of the actions and the referents of the names, as action names can be synonymous. The process of relating agents with specific actions can also help resolve some of the ambiguities from the agent elicitation step. For example, the identity of the 'Unidentified Loading Agent', referred to in Section 3.4.1, is reconciled when agents are identified with the 'Loading' action:

> Employees involved in the loading of the shuttle train: two positioners, two marshals, four chockers, one liaison agent and the driver of the bus taking Heavy Goods Vehicle drivers to the Amenity Coach. (1/2)

Data Flow Diagrams

SCS uses data flow diagrams (DFDs) (based on [DeM78]) as an initial 'first pass' technique to elicit, for each agent, the 'data flows' that they produce and consume. DFDs also help to identify the destination and source of these data flows. This information is then used as the basis for the action tabulation step. The language of DFDs is accessible and widely known. We found that DFDs of the system in the accident report provided a valuable visual overview of the flow of 'data' in the system. The 'data'

represented in these diagrams can be physical objects, information, or something less tangible, such as a command or other verbal instruction. For example, Figure 3.7 represents the following extract from Paragraphs 1 and 3 of the Channel Tunnel Safety Authority chapter:

> 1. The Channel Tunnel Safety Authority was established by the Governments in accordance with the provisions of the Treaty of Canterbury, signed on 12 February 1986 between the Republic of France and the United Kingdom of Great Britain and Northern Ireland, and relating to the construction and operation by private Concessionaires of a cross-Channel Fixed Link. The Intergovernmental Commission, also established under the Treaty, supervises in the name and on behalf of both Governments, all matters concerning the construction and operation of the Fixed Link. The Safety Authority (under Article 11 of the Treaty) advises and assists the Intergovernmental Commission on all matters concerning safety of the Fixed Link.
>
> ...
>
> 3. In the context of its duties the Safety Authority is charged in particular (under Article 11 (1) (c) of the Treaty) with examining reports concerning any incident affecting safety, making such investigations as are necessary and reporting thereon to the Intergovernmental Commission.

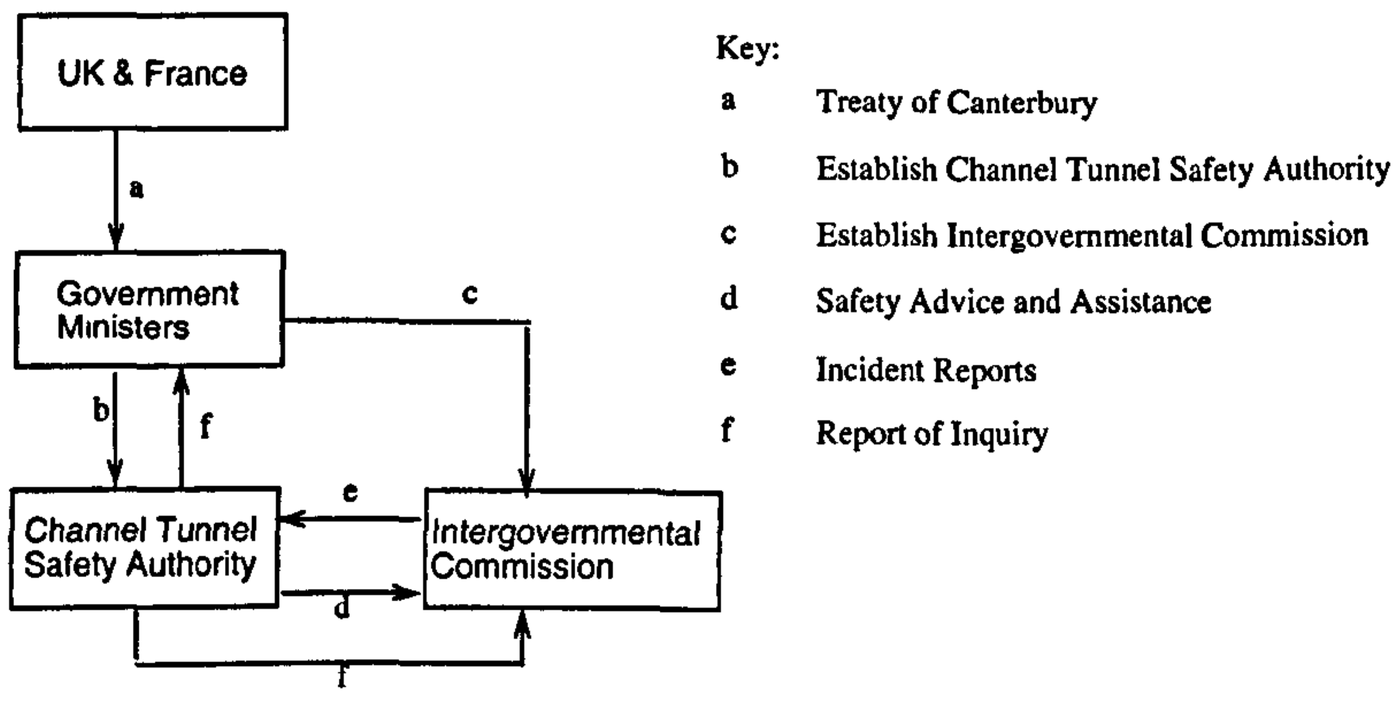

Figure 3.7: DFD of the 'Information Flow' Between the Governmental Entities

Action Tabulation

The information described in the DFDs can be used as the starting point for developing a table, for each agent, of the actions it performs. These tables are based on CORE's tabular collection forms and have a similar basic aim: to record all actions performed and the data produced or consumed. The layout of the tables emphasises how actions control the transition of information from the incoming data flows to the outgoing data flows. It also reflects the structure of EDAL action descriptions. These tables also

include the sources and destinations of each data flow (for example, see Figure 3.8). In requirements engineering, the tables are used to prompt and display what each agent 'does', as well as the pre- and post-conditions of the action. In accident report analysis, the tables also reveal information gaps in the report.

The tables are based on the CORE notion of calculating the 'transactions' of each 'viewpoint'. A 'transaction' is an action, its pre- and post-conditions, and the sources and destinations of these conditions. With SCS, transactions are calculated for the agent 'viewpoints', the leaf nodes of the hierarchy. For each agent the report is scanned completely for references to its actions. Given the problems with the size of the report, inconsistent referencing and the use of pronouns, this would be a time consuming and error prone task. For example, the Channel Tunnel fire report model has over 60 agents, implying over 60 passes of the report.

We chose to work through the report paragraph by paragraph, extracting each action. The report needs to be scanned only once. This action-oriented search can also be used to validate the agents in the hierarchy, as the performer of each action should already have been identified in the hierarchy.

Technical Experts

Reference	Source	Input	Action	Output	Destination
3/CTSA	Co-Rapporteurs	Incident Inquiry Information	Pass Incident Information	Incident Information	Co-Rapporteurs
2/Exec	CTSA	Unexamined General Design	Examine Safety of General Design	Approved General Design	CTSA
Assume 2,3, 4/Exec	CTSA	Unexamined HGV Design	Examine Safety of HGV Design	Unapproved HGV Design	CTSA

Figure 3.8: Action Table for Technical Experts

Descriptions of Actions

The descriptions of actions (to be distinguished from EDAL action descriptions) are an important reference point, both for the analyst producing the model and for the reader. Similar to agent descriptions, these are short natural language descriptions (for example, see Figure 3.9). They document the known information and assumptions about actions, which helps the analysts with the causal analysis step. They can also aid understanding of the model and associated proofs.

As with the agent elicitation steps, the action elicitation steps of the model face problems caused by both the natural language of the report and the slight incompatibility of the method with the accident report modelling application.

Establish Intergovernmental Commission The Treaty of Canterbury requires the establishment of an entity to monitor the construction and operation of the Channel Tunnel. This action establishes an intergovernmental commission to perform these tasks.

Figure 3.9: Extract from the Channel Tunnel Descriptions of Actions

Action Interpretation Difficulties

The natural language of the report, in particular the proliferation of synonyms, is problematic for action elicitation. For example, the following two sentences from Paragraph 2, Chapter 3, represent basically the same 'move' action being performed: 'the incident train left the platform', 'the incident train then moved off'. In this case, familiarity with the common definitions of these words makes resolving a general action name a straightforward task.

Where information is implicit or missing in the report, assumptions are required. However, in a number of cases the information given about the actions is inadequate for making educated assumptions without some additional domain knowledge. For example, 'in an emergency, the Chairman of the Channel Tunnel Safety Authority or his agent shall take the measures necessary for the safety of persons or property within the Fixed Link'. No further information is given on how 'emergency' or 'safety' are defined, or what the necessary measures might be.

The above sentence is presented in the report as an extract from the Treaty of Canterbury, and the terms have presumably been left purposefully vague in the treaty as the system was not yet implemented. However, without a more precise indication of what the terms mean in the current system, the above sentence adds little information to the accident report. If we wish to reason about whether these agents performed their duty, we have to assume what their duty was. As was found during agent elicitation, the existence of information gaps like these re-emphasise the need for a domain expert in the modelling process. However, the identification of such inadequate information in the report also demonstrates the value of using formal languages.

The representation of timing information has been highlighted as problematic in accident reports (see the section on 'temporal referencing' in the discussion of the weaknesses of accident reports in Chapter 1.1.7 and Johnson et al. [JMW95]). As the Channel Tunnel fire was an international incident, further problems could have been introduced by the different time zones used by the agents on either side of the tunnel. The behaviour in the report seems to have been described with reference to a single time zone, which makes modelling and understanding the accident report more straightforward. However, the report makes no reference to time zone differences, and this inevitably introduces inconsistencies between the report and the actual incident. Indeed, no reference is given in the report to which time zone is used.

Traceability

SCS action tables do not contain a reference to the source of the information they contain. In accident report analysis, this is quite simple to add, as the information has a fixed location in the report. An extra 'reference' column has been introduced to the action tables to contain this information (as can be seen in Figure 3.8). This column is also used to indicate when an assumption is made.

This information improves the traceability of the information in the model back to the report. This referential information is particularly important because the information represented in a single row of the table can be drawn from a number of locations in the report. This improved traceability can also be used to examine and justify assumptions.

Inconsistent Behaviour Descriptions

When the action tables of the sections of the Channel Tunnel fire report were combined to create a complete set of agent descriptions, there were, as would be expected, a number of inconsistencies. Most of these were due to synonymous action names and could be resolved easily, as we described with the 'move' example above. However, in some instances, the descriptions of the sources, destinations, or data flows differed. An example of this is the descriptions of the initial observations of the fire on the train:

> As the incident train passed, two security guards on duty in a building approximately 600m from the tunnel portal, saw a fire beneath a lorry on a wagon somewhere in the middle of the second rake of the incident train. They immediately informed their supervisor, who transmitted the information to the Terminal Control Centre in the French terminal. (3/3)
>
> One of the security guards that saw flames coming from the incident train before it entered the tunnel radioed this information to his supervisor, who in turn passed it to the Control Security Officer. From there it was passed to the French Terminal Control Centre, where it was received at 21.48 hrs. (3/8)

In the former quote, the information is relayed directly from the supervisor to the Terminal Control Centre. In the latter quote, the supervisor communicates with the Control Security Officer.

Such explicit inconsistent versions of events may have been elicited from conflicting views of different authors or witnesses. With no further information, it is not possible to resolve which, if any, of these versions of events is accurate. In these situations, we use the heuristic that, during transcription of a report, an error of omission is more likely than an error of commission.

The Value of DFDs

The differences between requirements engineering and accident report modelling result in a different role for DFDs. In requirements engineering, the nodes are easy to extract

using the agent hierarchy, and can then be used to prompt the elicitation of actions. In contrast, accident reports are explicit about the actions performed. After the actions are identified, they drive the completion of the other columns of the action table. As discussed above, this application also required a new approach to eliciting actions.

The construction of the action tabulations therefore takes place in conjunction with the construction of DFDs, rather than following it. Although the DFD construction could be omitted, we feel that it is beneficial as it produces an easy to understand overview of the information flow between the agents.

Representing Physical 'Data Flows'

The DFD and action tabulation steps view actions as data flow mechanisms. Problems arise when trying to describe 'data flows' of physical actions, such as the data flow that results from a train moving from one location to another. Modelling the train as a datum seems unreasonable, and there is no obvious 'destination' for whatever output this 'move' action produces.

Two approaches are identified to overcome this problem: directly passing physical information of entities between agents and extending the CORE environment agent to store physical properties of entities.

With the former option, modelling the movement of a train would involve passing the location information of the train as a direct input into agents whose actions depend on it. Although this maintains the traditional definition of the CORE environment agent, we felt that it creates a misleading overview of the system. For example, when the incident train passes through an area of the tunnel leaving smoke in its wake, the smoke would be viewed as a 'data flow' from the incident train to the train following it. The uppermost of the two diagrams in Figure 3.10 illustrates this.

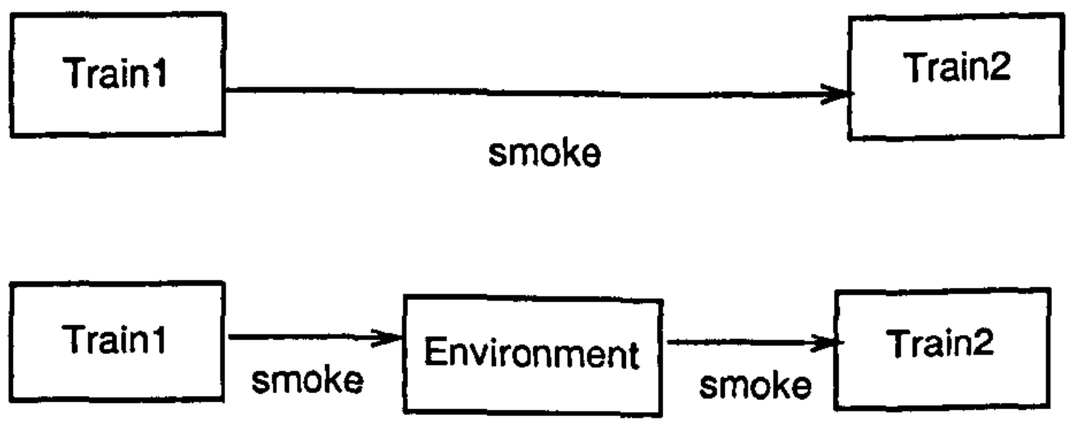

Figure 3.10: Extending the Environment Agent

We felt the indirect nature of the flow should be represented in the DFD and action tables. The lower of the two diagrams in Figure 3.10 illustrates this. This approach uses the environment agent to act as the destination of physical information, and thus as a global state for the physical properties of entities. For example, the location of the smoke is passed to the environment agent. The choice is one of style, but we feel the latter approach is better suited to modelling physical information.

Temporal Actions and Meta Actions

(E)DAL actions are atomic. However, the report describes some non-atomic actions, such as the allocation of lorries to the incident train:

> The allocation of the lorries to the incident train began at 21.19 hrs and was completed by 21.32 hrs.

As we are modelling accidents in retrospect, we can assume a level of description at which the actions are atomic. Alternatively, these actions can be represented by modelling the temporal meta-actions (begin, complete) as atomic actions (i.e., beginallocate, endallocate). In this case, the performance of the first action weakly obliges the second.

Iteration

Each step of the SCS method is iterated. Following the first iteration, the majority of the information in the action tables did not alter, although its structure did. As the analyst became more familiar with the entities and their properties, the entries in the action tables became better defined. One result of this is that many of the attributes and relations, which are modelled in the entity and relation elicitation steps described next, actually began to emerge during the action tabulation stage.

3.4.3 Adding Data Structures

After the identification of agents and actions, SCS focuses the analyst on eliciting the static aspects of the system, such as the non-agent entities, relations and functions.

The identification of the static aspects of the system is important to the model, as these affect the behaviour of the agents. In addition, it is important to identify areas of the report where insufficient or conflicting information is given about these static aspects.

Structuring Data Flows

SCS uses JSD data structure diagrams to decompose the complex data flows described in the DFDs. These are structured using sequencing, iteration, choice and non-determinism. However, there are few such complex data flows in the Channel Tunnel fire report. The level of detail in the Channel Tunnel fire report is generally too abstract for data flows to be analysed at a low level.

Simple data compositions are more common in the Channel Tunnel fire model, and are represented using a text list. For example, the data flow 'Evidence & Test Results' describes the set of information that the Eurotunnel management presented to the CTSA

to demonstrate the safety of the HGV system:

Evidence & Test Results:	RTMS Test Results + HGV Train Fire Development Test Results + Emergency Procedure Test Results + Tunnel Ventilation Test Results + Fire Detection System Test Results

The data decomposition step helps highlight instances where the report provides little detail about seemingly complex data flows. For example, although the report criticises the current alarm management system, which allows numerous alarm signals to overload the Engineering Management System operator, little further information is given about these signals. This criticism has clear consequences for human factors experts who want to prevent similar occurrences in other similar systems. Recommendation 30a of the report states: 'the Engineering Management System must be modified so that the operator is not faced with an unmanageable increase in workload during an emergency'. Without further detail, it is extremely difficult for the report reader to gain more than a general understanding of the problem and for the recommendations of the report to be useful in the redesign of the system, or the design of other similar systems.

Entity-Relationship-Attribute (ERA) Analysis

Entity-Relationship-Attribute (ERA) Analysis elicits the static structure of the model. The first stage of this process is identifying and presenting a comprehensive list of the entities of the system, their attributes, and the potential values of those attributes (see Figure 3.11).

SCS defines an *entity* as follows:

> Anything in the domain of discourse that performs or suffers actions, that has properties that change or are of interest, that enters into relationships with other entities that change or are of interest or anything that is a named individual of some significance.

An *attribute* is a property of an entity.

The action tables and data decompositions aid in the identification of some of the entities and attributes. Indeed this stage can be performed in conjunction with later iterations of action tabulation. In later iterations, the text contained in the action tables is more structured and formal, and the entities and attributes are more obvious. We extend the ERA analysis technique to enable explicit reference to the information source in the report.

The second product of ERA analysis is a textual equivalent of an Entity-Relation diagram. This also divides the relations into categories of association relations, instantiation relations and transformation relations. The association relations indicate predicates and functions. For example, the one-to-one relation 'drives' states that each driver drives only one train, and that train is driven only by that driver. Instantiation relations

Type	Entity	Attributes	Values	Reference
Agents	E_Staff	Training	{Yes, No}	19/8,...
	...	...	...	
Passive Entities	Treaty	Signed	{true, false}	1/CTSA
	Design	Status	{unfinished, unexamined, approved, unapproved}	2,3/Exec,...
	...	...	...	

Figure 3.11: Section of the Entity-Attribute List

identify potential structures of sorts. For example, an instantiation of a train ID is an instantiation of an integer. Transformation relations identify functions that transform a number of objects into one of another sort, or vice versa [PFAB86]. For example, the information on the Channel Tunnel fire accrued from different agents is transformed into the report source material by the Co-rapporteurs. The E-R table for these examples is given in Figure 3.12.

Type	Relationship	From	To
Association			
	Drives (1:1)	Driver ID	Train ID
	...	...	...
Instantiation			
	Is a	Train ID	Integer
	...	...	...
Transformation			
	Becomes (n:1)	IncidentInfo	ReportSource
	...	...	...

Figure 3.12: Example Instances of the Three Types of Relation

3.4.4 Causal Analysis

The causal analysis step describes the actions of each agent with respect to the conditions of occurrence and their effects. In addition, disabling factors for each action are described.

The enabling factors for each action are drawn from the input column in the action tabulation (and any associated data decomposition diagrams). Where the disabling factors of an action are explicit within the report, they are added directly into the causal analysis. In general, these factors are implicit, but common sense and knowledge of the scope of the model can be used to generate other disabling factors.

For example, the scope of the Channel Tunnel fire model only covers the construction of one set of design plans of each type (HGV and general). We therefore assume that a design should be produced only if the design is unfinished. If the design is

finished, we assume that another design of the same type should not be produced. The causal action description for the *Produce_Design* action performed by the 'designers and constructors' agent is defined in Figure 3.13 (*e_man* is the agent representing the Eurotunnel management).

Produce Design: $DESIGN \times DESIGN\_SAFETY$:
Enablement: status(D,unfinished)
Disablement: ¬(Enablement)
Effect: Produces design with specific safety level.
Unexamined design can be passed to *e_man*
for permit application.

Figure 3.13: Causal Action Description from the Channel Tunnel Report

The action is permitted if the status of the design is unavailable (i.e., has not been finished). Once performed, the action is prohibited from being performed again for that design. Performing the *Produce_Design* action also determines the safety of the design.

We use a tabular technique that extends the SCS technique in that it provides information, where applicable, on the prescriptor, the action that incurs the prescriptive information, the agent affected, the prescripted action, and the type of prescription. This is exemplified in Figure 3.14, where the Channel Tunnel Safety Authority is represented by the term 'CTSA' and the Intergovernmental Commission is represented by the term 'IGC'.

Agent	Performs Action	Affects Agent	Deontic Effect	Affects Action
UK &	Sign(cantTreaty)	Govt. Ministers	IO	Establish(CTSA)
France	Sign(cantTreaty)	Govt. Ministers	IO	Establish(IGC)
...	...	...	...	...

Figure 3.14: Prescriptive Behaviour Table Extract

3.4.5 Constructing the Formal Model

Language Definition

Earlier, we discussed the importance of highlighting the agents, actions and entities mentioned in the accident report. The language definition section of the formal model can be used as a concise index of the agents, actions and entities involved in the formal model. Previous sections have shown how the size and layout of natural language reports makes it very difficult to extract this information. The concise nature of the language definition section reduces the likelihood of multiple definitions and synonyms, and also enables the validity of the items in the index to be judged by others.

EDAL's use of an agent meta-sort means that many different sorts of agent can exist in the model. This is well suited to the agents typically found in accident reports, which are generally extremely diverse.

The action declarations are extracted from the action tables. Similarly, the agent declarations can be extracted from the agent hierarchy and entities from the entity lists.

Entity attributes and relations are used to identify functions and predicates. Additional data sorts are defined to model the values of attributes. For example, a sort $DESIGN\_STATUS$ is defined for the status attributes of the $DESIGN$ entity.

EntitySorts
DESIGN_STATUS
DESIGN_SAFETY
DESIGN
...
Constants
unfinished, unexamined, approved, unapproved: DESIGN_STATUS
unavailable, none, well_designed_reliable_equipment,
clear_effective_procedures, both: DESIGN_SAFETY
hGVDesign, genDesign: DESIGN
...
Entity Variable
D: DESIGN

Figure 3.15: Channel Tunnel Language Definition Extract

Model Axioms

It is straightforward to translate action descriptions and prescriptions from the action tables and prescriptive behaviour tables. However, the immediacy of immediate obligation introduced some problems during the formalisation. Consider the outcome of signing the treaty, as described in Figure 3.14. Two actions are immediately obliged to be performed by the Government Ministers, thus the agent cannot perform an action that leads to a normative scenario. This is not the intention of the prescription, and we altered the formal model to define these immediate obligations as occurring in (an arbitrary) sequence:

$$[UF, Sign(T)]signed(T)$$
$$signed(cantTreaty) \wedge \neg established(ctsa) \rightarrow IO(gm, Establish(ctsa))$$
$$signed(cantTreaty) \wedge established(ctsa) \wedge \neg(established(igc)) \rightarrow$$
$$IO(gm, Establish(igc))$$

The modal operator of EDAL has a number of important features. The square brackets make action descriptions easily distinguishable from the static information in the model. By pairing agents and actions, the agent performing the action is made incontrovertible, unlike in the similar logics of Meyer [Mey88] and Fiadeiro and Maibaum [FM91]. For example, compare the ambiguity concerning the identity of the ordering agent in $[order(A, B)]\phi$ with its EDAL counterpart $[B, order(A)]\phi$. By examining the action descriptions in an EDAL model, it is possible to easily determine which agents can perform which actions, and whether the same action has the same effect when performed by different agents.

The action description introduces the notion of sequence without the complex machinery of real-time logics. Given the limited evidence in the aftermath of an accident it is, typically, impossible to reconstruct complete timelines for all agent actions. Furthermore, the precision of the timing information in reports is variable and the timepieces used for reference by the report may also be inaccurate (as in the King's Cross fire report [Fen88]). The modal connective can be used to represent the effects of actions performed at both known and unknown time intervals [BJT97].

For added clarity in the formal model, Potts et al. [PFAB86] recommend separating prescriptive statements from action descriptions. The separation requires the introduction of additional predicates. For example, with a statement such as $[A, \alpha]P(B, \beta)$, a predicate ϕ is introduced to denote the effect of A performing α, $[A, \alpha]\phi$. An implication is then defined: $\phi \rightarrow P(B, \beta)$. This approach enables the prescriptive behaviour and descriptive behaviour to be more easily distinguished. It also makes the specification more uniform, as these additional predicates are already required to describe situations where an obligation or permission results from more than one action. For example, the technical experts are weakly obliged to examine the HGV design safety only as a combination of a design having been produced and the Channel Tunnel Safety Authority having arranged for this scrutiny:

$$
\begin{aligned}
&[CT, Arrange\_Scrutiny(D)]arrangedScrutiny(D)\\
&[DC, Produce\_Design(D, S)]\\
&\quad status(D, unexamined) \wedge safety(D, S)\\
&\ldots\\
&arrangedScrutiny(D) \wedge status(D, unexamined) \rightarrow\\
&\quad IO(TX, Examine\_Safety(D))
\end{aligned}
$$

This separation also helped identify a number of problems in the model. As an example, a decision made during the formal modelling step to separate the $DESIGN$ attributes of the level of concern about the design ($DESIGN\_CONCERN$) and the status of the design ($DESIGN\_STATUS$), led to some confusion in the causal analysis tables and the formal model. In grouping the related axioms together, it was clear that the some axioms for the *status* predicate still expected $DESIGN\_CONCERN$ values.

A slight disadvantage of this heuristic is that the added predicates have to be explicitly negated by the model, to prevent them reasserting an obligation/permission once the obligation/permission has been removed.

The axioms in the Channel Tunnel fire model are further grouped by agent and order of occurrence. They are also split between axioms that define generic system properties and those that define deterministic behaviour, as described in the report. We found that all the groupings, apart from the last, make the axioms easier to locate in the model. We found the final grouping useful as it highlights the degree to which the model is determined and the gaps that need to be filled to produce a general model.

3.5 Model Construction Issues

As well as facilitating a relatively smooth transition from the natural language of the accident report to the formal model, the steps of the method each provide new insights into the report. Each step therefore has analytical rewards in addition to its use for elicitation and representation.

As each step focuses on a particular aspect of the model, certain 'types' of error associated with that aspect are uncovered. These problems can then be addressed without the distraction of other 'types' of errors. For example, agents are identified, structured, and described early on. During the action tabulation step, potentially problematic and distracting agent-based issues, such as identifying who performed actions in passive sentences, have thus already been resolved.

The multiple steps of SCS introduce some redundancy and scope for inconsistency between steps. The use of natural language enables some ambiguity to remain. However, the final step of producing a formal specification helps ensure that the ambiguity is removed.

3.5.1 Changing the Method Steps

Although the steps of the SCS were adaptable to accident report modelling, there may be alternative techniques in existence that would fit into the method and would be better suited to this application. In particular, DFDs are unsuited to modelling the physical effects of actions and changing the accepted conventions of the technique may confuse the reader. Instead, it may be better to omit the use of DFDs from the method.

In the work presented, the final formalisation step of SCS is targeted on EDAL. However, modifying the method for different target formalisms is possible. The greater the similarities between the syntactic categories of the new target formalism and MAL, the fewer modifications should be necessary [PFAB86]. Significant benefits, in terms of consistency and efficiency, would be achieved if the output of the SCS semiformal steps could also be used to build formal models in other formalisms to examine other

aspects of the model. However, none of the formalisms used to model accidents feature all of the syntactic categories of EDAL and some modification would probably be necessary. SCS could also be used to build accident report models with related deontic logics, such as those of Meyer [Mey88] and Fiadeiro and Maibaum [FM91].

3.5.2 Tool Support

As mentioned earlier, the use of recognised techniques in the steps of SCS improves the likelihood of tool support being available. However, there are several further ways that tools could aid the model construction.

Elicitation Support

Eliciting the agents, actions, and entities from the full report is extremely time consuming. Further time is required to ensure the correct interpretation from the given context. A number of tools from the fields of information retrieval and text manipulation were examined in the hope that they may be able to assist with the basic elicitation task.

Concordance tools, such as Concorder [Ran97], produce a distribution of the words in the document. Document analysis programs, such as Wordsmith [Sco97], search the document for noun phrases. With filtering, either tool can be used to highlight the names given to agents and entities in an electronic copy of an accident report. However, none of the tools examined provided any straightforward means to reference the context of the terms' usage in the report. Webcrawlers are a possible solution, as they can store hypertext links to particular textual locations in a document, although the text needs to be marked up before the webcrawler can produce links to specific locations.

Our survey of existing tools was unsuccessful in finding an easily adaptable tool. However it did help clarify a realistic definition of the properties sought in the tool: given a document, the tool constructs a list of the nouns and verbs in the document, with hypertext links to each instantiation in a marked up version of the document. The hypertext links enable swift checking of the local context of the term.

Consistency Support

As described earlier, some improvements were made to the traceability of information in SCS. However, a more sophisticated ability to handle change management between the steps is desirable. This is one of the features of many current requirements engineering tools, such as Quality Systems and Software's Dynamic Object-Oriented Requirements System (DOORS) 4 and Rational Software's Requisite Pro.

The International Council on Systems Engineering (INCOSE) has produced a set of evaluation criteria for requirements tools that includes semiautomatic requirements identification, traceability, maintaining consistency, change history and version control, and rationale capture [MF96]. As SCS uses standard methodological steps, any

methodologically flexible INCOSE approved tool should bring significant improvements to traceability and consistency in the EDAL modelling process.

Formalisation Support

The EDAL specification step itself does not use a well-known technique like some of the other steps, and there are no existing tools to assist with this step. An editor to enforce the EDAL syntax would be a valuable tool. This could be adapted from Booth's MAL parser [Boo87].

3.5.3 Scope Issues

In constructing the Channel Tunnel fire report model, we used a wide scope approach for the first steps. The size of the report made it difficult to perform the elicitation tasks on the full document. As described, the report was partitioned, and each section was modelled separately.

After the initial few steps in the construction of the model had been performed, and the information was in a semiformal state, the majority of the errors, ambiguities and inconsistencies found had been uncovered. To reduce the size and construction time of the formal model, a hypothesis driven approach to accident report analysis was adopted for the final steps of the method.

As discussed earlier, a hypothesis driven approach, such as the conclusion validation approach, produces a less generally applicable model, but it is smaller, and thus easier to examine and less prone to errors in construction and reasoning. In addition, by performing the initial construction steps using the wide scope approach, much additional information was revealed and many further omissions and errors were uncovered. The accuracy of the final model was thus improved.

3.5.4 Rhetoric Issues

Rhetorical devices can bias the interpretation of accident reports [SJ98]. The use of formal methods and techniques does not remove all bias or rhetoric. Indeed, the additional levels of interpretation and assumption potentially add further bias to the model. As with other approaches, the purposes of the analysis, the method used, and the perspective of the analyst may affect the model produced [Lek97].

However, the use of formal methods helps clarify and correct the argumentation and identify inconsistencies and ambiguities in the report. This in turn can reduce or remove the effects of some of the rhetorical devices commonly found in reports (as identified by Snowdon and Johnson [SJ98]): argumentation with partial evidence, bias through repetition, the use of indirect evidence, and typographical emphasis. Furthermore, a disciplined approach to modelling, in which all assumptions of incomplete

and ambiguous information are documented and justified, facilitates the identification of analyst bias in the model. As discussed in Chapter 1.2.6, the wide scope approach should also help reduce confirmation bias in the model.

3.5.5 Expertise Issues

There is agreement in the formal methods community that successful application of formal methods relies on the analyst having a good knowledge of the application domain and the method being targeted at the domain [LG97, BBD+96]. EDAL and the adapted SCS method are targeted on accident report modelling. However, successful application still requires knowledge of the system by the analyst. In this thesis, the unfamiliarity of the author with the systems represented in the accident reports, and with further detail of the accident, is a factor in the accuracy of the models produced. Further knowledge would have permitted more assured assumptions. On the other hand, unfamiliarity may also have prevented implicit assumptions being made about the report, based on expert knowledge, thus resulting in a more faithful model of the information in the report itself. A team of analysts and domain experts, such as discussed in Chapter 1.2.3, would gain both these benefits [ELC+98]. The difficulties encountered during our analysis of the accident report reflect the difficulties faced by other safety professionals, who are familiar with accident reports but not necessarily with the system being described.

3.5.6 Evaluation Issues

This application of SCS demonstrates that it can be applied to modelling real accident reports. We have also highlighted the perceived benefits and shortcomings of the method for this application. However, the application was performed without the constraints of an industrial setting, such as strict time and budget constraints. Although the steps of the method are well known, there has been no evaluation of the ease of use of the method in this application for analysts or domain experts. Similarly, claims about the readability, scalability and traceability require more rigorous validation. The successful application presented here can be viewed as a proof of concept and work should now begin on a more rigorous feasibility study of the use of the method in an industrial setting.

3.6 Summary

This chapter outlines the requirements engineering method SCS as a method suited to accident report modelling. We first discussed the similarities between requirements engineering and accident report modelling, and described the benefits of using work from the former established research field to inform progress in the relatively new field of accident report modelling. The motivation for choosing SCS was that it was

designed for eliciting the syntactic categories of EDAL and that it guides and organises the activity of the analyst.

Using the Channel Tunnel fire report as a case study, we examined each step of the method and discussed the alterations to the method that improve its suitability to the application. We presented the difficulties involved in modelling a large accident report and described how each of the problems was overcome. We also included a discussion of tools that could improve the quality and efficiency of the modelling process.

In the following chapter we outline a formal model of the behaviour described in the Channel Tunnel report, including the setup of the regulatory framework, and the design and implementation of the system designs.

Part III

Using the Model

Chapter 4

Outline of an Accident Report Model

This chapter outlines the axioms of a formal model of the Channel Tunnel fire report. These are then used to prove theorems in the chapter following. The scope of the model covers the behaviour from the signing of the Treaty of Canterbury in 1986 and the design of the system designs to the performance of a test exercise on the 10th of November 1988.

To preserve space, the language definition and unused axioms of the model are omitted here, but can be found in Appendix E. In some cases, the exact order of events in the system is not known. In the theorems that follow, these situations have no bearing on the outcome (for example, the order in which the HGV and general design permits were applied for by Eurotunnel). In such instances, we assume an arbitrary action trace.

Before introducing the functional description of the system, we introduce a set of constraints, based on the non-functional requirement of an 'Acceptable Level of Safety' (*ALoS*). These constraints are axioms of the system and represent functional translations of how the *ALoS* requirement applies to different agents.

4.1 Defining an Acceptable Level of Safety

> **Quote 1:** The objective of the designers, constructors, operators and regulators of a transport system must be to ensure that it provides an acceptable level of safety for the staff and users of the system. Such a level of safety depends upon well-designed, reliable equipment operated in accordance with clear and effective procedures, particularly in the management of emergencies. (1/Exec)

The 'Acceptable Level of Safety' (*ALoS*) constraint applies to a number of entities (the designers, constructors, operators and regulators). In the model, the operators are taken

to be the Eurotunnel management and the regulators are taken to be the Intergovernmental Commission and the Channel Tunnel Safety Authority.

Designers and Constructors

The designers and constructors of the system are not referred to elsewhere in the report. However, their duty is quite plain from the above paragraph: to ensure the equipment is well designed and reliable, and to ensure the procedures are clear and effective.

There are two designs referred to in the development of the Channel Tunnel system. The general design defines the equipment and procedures to support freight, passenger, and tourist shuttles. The HGV design defines the equipment and procedures to support the HGV shuttles. The two designs are modelled by the constant values $hGVDesign$ and $genDesign$ of the $DESIGN$ sort. An attribute of a design is safety. The $DESIGN\_SAFETY$ sort has the constant values:

$$unavailable, none, well\_designed\_reliable\_equipment,$$
$$clear\_effective\_procedures, both : DESIGN\_SAFETY$$

The $unavailable$ value holds if the design has not finished production. The $both$ value holds if the design has both well designed, reliable equipment and clear, effective procedures. The predicate $safety$ takes a value of sort $DESIGN$ and a value of sort $DESIGN\_SAFETY$.

The single agent representing the designers and constructors is dc of sort D_C. The property $dcALoS$ defines the role of the designers and constructors in ensuring an acceptable level of safety. $dcALoS$ implies that, for all designs, the safety of the design is either unavailable, because the design has not been finished, or has both well-designed, reliable equipment and clear and effective procedures:

$$dcALoS \rightarrow$$
$$\forall D : DESIGN.(safety(D, both) \vee safety(D, unavailable)) \qquad (4.1)$$

Channel Tunnel Safety Authority

The Channel Tunnel Safety Authority 'advises and assists the Intergovernmental Commission on all matters concerning safety of the Fixed Link'(1/CTSA). From the following paragraphs, we assume that part of this role is to approve the designs of the systems and procedures before they are introduced:

> **Quote 2:** Concerning the safety of the Eurotunnel system overall, approval was given [by the Channel Tunnel Safety Authority] to engineering, equipment and rolling stock designs only after much scrutiny by appropriate specialists and the

production of a detailed safety case. Railway services were introduced progressively in a prolonged commissioning process in which operating procedures were developed and staff gained experience of a unique transportation system. (2/Exec)

Quote 3: The Channel Tunnel Safety Authority deliberated at great length before it was in a position in 1993 to recommend to the Intergovernmental Commission that no objection should be made on safety grounds to the Heavy Goods Vehicle shuttle design and its operating procedures. (3/Exec)

Additionally, we can see that if there are problems perceived in the designs, the Channel Tunnel Safety Authority expresses its concerns:

Quote 4: As early as 1992, the Channel Tunnel Safety Authority expressed concern about the possible size of a fire on a Heavy Goods Vehicle shuttle at the time of detection, the rate of fire development and spread, the size of a fire able to overcome the Supplementary Ventilation System, and the conditions concerning temperature, visibility and toxicity which would allow safe evacuation of passengers and crew into the Service Tunnel. (3/Exec)

In this case, the Eurotunnel management reacted to these concerns:

Quote 5: At the time, Eurotunnel put forward a substantial amount of evidence, backed up by an extensive programme of experiments and tests, to allay the Channel Tunnel Safety Authority's concerns. (4/Exec)

Once the Eurotunnel management had presented the evidence to the Channel Tunnel Safety Authority, the Authority's concerns were laid to rest:

Quote 6: On the basis of this evidence, the Channel Tunnel Safety Authority recommended to the Intergovernmental Commission in 1994 that a favourable response should be made to the request of Eurotunnel to the start of the Heavy Goods Vehicle shuttle service. (5/Exec)

The level of concern about a particular design is modelled as an attribute of that design. The $DESIGN\_CONCERN$ sort has the constant values:

$$addressed, unaddressed, none : DESIGN\_CONCERN$$

The predicate *concern* takes a value of sort $DESIGN\_CONCERN$ and a value of sort $DESIGN$. The Channel Tunnel Safety Authority's approval of a design is denoted by the predicate $ctsaApproval$, which takes a value of sort $DESIGN$.

The agent representing the Channel Tunnel Safety Authority is $ctsa$ of sort $CTSA$. The property $ctsaALoS$ defines the role of the Channel Tunnel Safety Authority in ensuring the procedures and equipment have an acceptable level of safety. $ctsaALoS$ implies that, for all designs, the approval of a design implies that there are both no unaddressed concerns and that the design includes both well designed, reliable equipment and clear, effective procedures:

$$
\begin{aligned}
&ctsaALoS \rightarrow \\
&\quad \forall D : DESIGN.(ctsaApproval(D) \rightarrow \\
&\qquad (\neg(concern(D, unaddressed)) \wedge safety(D, both)))
\end{aligned} \tag{4.2}
$$

Eurotunnel Management

The report implies that the Eurotunnel management did not ensure an acceptable level of safety:

> **Quote 7:** A thorough analysis however, set out in this Inquiry Report, raises important questions about the overall safety of the system in these and similar circumstances, and has brought to light faults and weaknesses that must be corrected. Before commercial service may be resumed at its former level, all of these questions must be answered by Eurotunnel to the satisfaction of the Channel Tunnel Safety Authority. (6/Exec)

We have argued above that other agents in the system are responsible for ensuring an acceptable level of safety in the system designs. We assert that the Eurotunnel management is responsible for implementing these safe designs properly.

More precisely, we assume that Eurotunnel management ensures an acceptable level of safety by using only designs that the Intergovernmental Commission has permitted, by training the Eurotunnel staff and by introducing the relevant operating procedures. The training of the staff is prominently featured in the report as the responsibility of the management. Although it is not explicitly linked with ensuring an acceptable level of safety, experienced staff heighten the level of safety for themselves and the customers [Rea97]. The latter part of Quote 2 above seems to imply that the training of the staff and the introduction of procedures was obliged as part of the permit being granted. The only procedures explicitly mentioned in the report text are the dangerous goods policy (35-37/1) and the policy for ensuring a minimum separation between trains (2/3). These apply only to the HGV design and are intended to improve system safety.

The agent representing the Eurotunnel staff is e_staff of sort E_STAFF. The level of training of the staff is a binary attribute of this agent. It is modelled by the predicate *trained*, which takes a value of sort E_STAFF. The level of training and the level of experience of the staff could be modelled separately. However, such a distinction is not germane to the aspects of the accident we are modelling. We therefore used the *trained* predicate to denote both.

The dangerous goods policy and the policy for ensuring a minimum separation between trains are denoted by the constant values $dgPolicy$ and $minSepPolicy$ of sort $POLICY$. The predicate *implemented*, which takes a value of sort $POLICY$, denotes whether or not a policy has been implemented by the Eurotunnel management.

The agent representing the Eurotunnel management is e_man of sort E_MAN. The property $emanALoS$ defines the role of the Eurotunnel management in ensuring the

procedures and equipment are implemented with an acceptable level of safety. In the case of the general design permit being awarded, $emanALoS$ implies that following the next action performed by the Eurotunnel management, the staff should have been trained. In the case of the HGV design permit being awarded, $emanALoS$ implies that following the next three actions performed by the Eurotunnel Management, the staff should have been trained, and the dangerous goods policy and the policy for ensuring a minimum separation between trains should have been implemented:

$$\begin{aligned}
&emanALoS \rightarrow \\
&\quad (permit(hgv) \rightarrow \\
&\qquad \forall \alpha, \beta, \gamma : Act.[e_man, \alpha][e_man, \beta][e_man, \gamma] \\
&\qquad\quad (trained(e_staff) \wedge implemented(minSepValue) \wedge \\
&\qquad\quad implemented(dgPolicy)) \\
&\quad \wedge (permit(gen) \rightarrow \\
&\qquad \forall \alpha : Act.[e_man, \alpha] trained(e_staff)))
\end{aligned} \tag{4.3}$$

4.2 Axioms of the Model

Designers and Constructors

The first deontic axiom for the designers and constructors of the Channel Tunnel system states that if a design is unfinished, they are permitted to produce a design with both well-designed, reliable equipment and clear and effective procedures (Clause 4.4). The second deontic axiom states that they are prohibited from producing a design that does not have both these properties (Clause 4.5). If the design has been finished, we assume that the permission to produce that design is revoked (Clause 4.6). If the report is unfinished, the safety level is unavailable (Clause 4.7). A finished design has an available level of safety (Clause 4.8):

$$status(D, unfinished) \rightarrow P(DC, Produce\_Design(D, both)) \tag{4.4}$$

$$\begin{aligned}
&status(D, unfinished) \rightarrow \\
&\quad \neg P(DC, Produce_Design(D, S)) \wedge \neg(S = both)
\end{aligned} \tag{4.5}$$

$$\neg status(D, unfinished) \rightarrow \neg P(DC, Produce\_Design(D, S)) \tag{4.6}$$

$$status(D, unfinished) \rightarrow safety(D, unavailable) \tag{4.7}$$

$$status(D, DS) \wedge DS \neq unfinished \rightarrow safety(D, S) \wedge S \neq unavailable \tag{4.8}$$

Performing the action *Produce_Design* has the effect of altering the status of a design to 'unexamined'. The safety level of the design is also asserted in the predicate *safety*:

$$[DC, Produce\_Design(D, S)] status(D, unexamined) \wedge safety(D, S) \tag{4.9}$$

Signing the Treaty

The next axioms we address concern the signing of the Treaty of Canterbury (modelled by the constant *cantTreaty*), as described in 1/CTSA:

> **Quote 8:** The Channel Tunnel Safety Authority was established by the Governments in accordance with the provisions of the Treaty of Canterbury, signed on 12 February 1986 between the Republic of France and the United Kingdom of Great Britain and Northern Ireland, and relating to the construction and operation by private Concessionaires of a cross-Channel Fixed Link. The Intergovernmental Commission, also established under the Treaty, supervises in the name and on behalf of both Governments, all matters concerning the construction and operation of the Fixed Link. The Safety Authority (under Article 11 of the Treaty) advises and assists the Intergovernmental Commission on all matters concerning safety of the Fixed Link.

This treaty initiates the setting up of the regulatory frameworks that should ensure an acceptable level of safety. The assumption is made that signing this treaty was permitted (Clause 4.10). Furthermore, we assume that once signed, the permission to sign a treaty is revoked (Clause 4.11):

$$\neg signed(T) \rightarrow P(UF, Sign(T)) \tag{4.10}$$

$$signed(T) \rightarrow \neg P(UF, Sign(T)) \tag{4.11}$$

The action description for the *Sign* action is straightforward:

$$[UF, Sign(T)]signed(T) \tag{4.12}$$

This *signed* predicate has the effect of obliging the establishment of the Channel Tunnel Safety Authority and Intergovernmental Commission (Clauses 4.13 and 4.14). As discussed earlier, these obligations are incurred in an arbitrary order:

$$signed(cantTreaty) \wedge \neg(established(ctsa) \rightarrow IO(gm, Establish(ctsa)) \tag{4.13}$$

$$signed(cantTreaty) \wedge established(ctsa) \wedge \neg(established(igc)) \rightarrow IO(gm, Establish(igc)) \tag{4.14}$$

The action descriptions for the *Establish* action are again straightforward (Clauses 4.15 and 4.16):

$$[G, Establish(CT)]established(CT) \tag{4.15}$$

$$[G, Establish(I)]established(I) \tag{4.16}$$

Applying for Permits

The duty of the Intergovernmental Commission is to supervise the construction and operation of the tunnel:

$$established(igc) \rightarrow igcSupervise \tag{4.17}$$

The assumption is made that, once the designs are completed, the Eurotunnel management must send them to the Intergovernmental Commission for scrutiny before a permit is granted. The report makes it clear that the designs are sent for scrutiny, but does not make it clear whether the designs would be sent directly to the Channel Tunnel Safety Authority or via the Intergovernmental Commission. We assume the latter, as the Channel Tunnel Safety Authority's role is one of assisting the Intergovernmental Commission and the Intergovernmental Commission has other concerns, as well as safety, over the designs of the system. Paragraph 1 of the Channel Tunnel Safety Authority section states that the Intergovernmental Commission supervises 'all matters concerning the construction and operation of the Fixed Link'. The permit application process is modelled by the following axioms:

$$\begin{aligned}&(status(D, unexamined)) \rightarrow \\ &\quad O(E, Apply\_Permit(D)) \wedge \\ &\quad P(E, Apply\_Permit(D))\end{aligned} \tag{4.18}$$

$$[E, Apply\_Permit(D)]permitApplied(D) \tag{4.19}$$

The Intergovernmental Commission then sends the system designs to the Channel Tunnel Safety Authority for safety assessment:

$$\begin{aligned}&igcSupervise \wedge permitApplied(D) \wedge \neg sentAssess(D) \rightarrow \\ &\quad IO(igc, Safety\_Assess(D))\end{aligned} \tag{4.20}$$

$$[I, Safety\_Assess(D)]sentAssess(D) \tag{4.21}$$

Assessing the Designs

The duty of the Channel Tunnel Safety Authority is described as advising and assisting the Intergovernmental Commission on safety issues:

$$established(ctsa) \rightarrow safetyAdvise(ctsa, igc) \tag{4.22}$$

We assume that as part of this duty, the Channel Tunnel Safety Authority arranges for the designs sent for safety assessment to be examined by technical experts, as implied by Quote 2:

$$safetyAdvise(CT, I) \wedge sentAssess(D) \wedge \neg arrangedScrutiny(D) \rightarrow$$
$$IO(CT, Arrange\_Scrutiny(D)) \quad (4.23)$$
$$[CT, Arrange\_Scrutiny(D)]arrangedScrutiny(D) \quad (4.24)$$

This assessment is crucial to the Channel Tunnel Safety Authority ensuring an acceptable level of safety. Rule 4.23 is the first of a number of rules that represent the regulatory and managerial safeguards that exist in the system. Using the deontic operators to model these safeguards facilitates analysing if, how, and when they are broken.

Assessment Results

In a more generic model, the assessment of the technical experts could be positive or negative for either design. However, in the report only the general design passed scrutiny, as described in Quote 2 given above:

$$arrangedScrutiny(D) \wedge (status(D, unexamined)) \rightarrow$$
$$IO(tech\_ex, Examine\_Safety(D)) \quad (4.25)$$
$$[TX, Examine\_Safety(gen)]$$
$$status(gen, approved) \wedge concern(gen, none) \quad (4.26)$$

The HGV design took longer to be approved, as described in Quotes 3-6. Paragraph 4/Exec enumerates the concerns of the Channel Tunnel Safety Authority:

Quote 9: This addressed, in particular:

- the ability of the fire detection system to detect a fire early enough for evacuation to proceed safely.
- the rate of fire development in a Heavy Goods Vehicle.
- the ability of the tunnel ventilation systems to control the air flow in the vicinity of a fire.
- the effectiveness of the procedures for dealing with an emergency in the tunnel, including the evacuation of the incident train and the movement of other trains in the Tunnel.
- the ability of the Rail Traffic Management system to maintain, in all circumstances, the necessary minimum separation distance between Heavy Goods Vehicle shuttles and other trains.

We assume the concerns raised by the Channel Tunnel Safety Authority over the HGV design were the result of a similar process of scrutiny as that undergone by the general design (Clause 4.27). We also assume that the Eurotunnel management have permission to perform all the tests and experiments (Clause 4.30):

$$[tech\_ex, Examine\_Safety(hgv)]$$
$$status(hgv, unapproved) \wedge concern(hgv, unaddressed) \quad (4.27)$$
$$concern(D, unaddressed) \wedge \neg(announcedConcern(D)) \rightarrow$$
$$IO(CT, Express\_Concerns(D)) \quad (4.28)$$
$$[CT, Express\_Concerns(D)]announcedConcern(D) \quad (4.29)$$
$$true \rightarrow$$
$$(P(e\_man, Request(cfds, Reaction\_Test)) \wedge$$
$$P(e\_man, Request(rtms, Separation\_Test)) \wedge$$
$$P(e\_man, Request(tester, HGV\_RoF\_Dev\_Test)) \wedge$$
$$P(e\_man, Request(ems, Vent\_Test)) \wedge$$
$$P(e\_man, Request(tester, Em\_Proc\_Test)) \quad (4.30)$$

The test actions given below could be modelled at a higher level of abstraction. We chose this lower level of abstraction, as we want to outline the number and diversity of the tests performed by the Eurotunnel Management. Little detail is given about the results of the tests, other than that they convinced the Channel Tunnel Safety Authority that their concerns had been addressed (Clause 4.48):

$$[e\_man, Request(cfds, Reaction\_Test)]testCfds \quad (4.31)$$
$$testCfds \rightarrow IO(CF, Reaction\_Test) \quad (4.32)$$
$$[CF, Reaction\_Test]testedCfds \wedge \neg testCfds \quad (4.33)$$
$$[e\_man, Request(rtms, Separation\_Test)]testRtms \quad (4.34)$$
$$testRtms \rightarrow IO(RT, Separation\_Test) \quad (4.35)$$
$$[RT, Separation\_Test]testedRtms \wedge \neg testRtms \quad (4.36)$$
$$[e\_man, Request(tester, Hgv\_RoF\_Dev\_Test)]testFireDev \quad (4.37)$$
$$testFireDev \wedge testedProc \rightarrow IO(TES, Hgv\_RoF\_Dev\_Test) \quad (4.38)$$
$$[TES, HGV\_RoF\_Dev\_Test]testedFireDev \wedge \neg testFireDev \quad (4.39)$$
$$[e\_man, Request(tester, Test\_Em\_Proc)]testProc \quad (4.40)$$
$$testProc \rightarrow IO(TES, Em\_Proc\_Test) \quad (4.41)$$
$$[TES, Em\_Proc\_Test]testedProc \wedge \neg testProc \quad (4.42)$$
$$[e\_man, Request(ems, Test\_Vent)]testVent \quad (4.43)$$
$$testVent \rightarrow IO(EM, Vent\_Test) \quad (4.44)$$
$$[EM, Test\_Vent]testedVent \wedge \neg testVent \quad (4.45)$$
$$testedCfds \wedge testedRtms \wedge testedVent \wedge testedFireDev \wedge testedProc \rightarrow$$
$$testResults \quad (4.46)$$
$$announcedConcern(hgv) \wedge testResults \rightarrow$$
$$P(e\_man, Present\_Evidence(hgv)) \quad (4.47)$$
$$[e\_man, Present\_Evidence(hgv)]$$
$$status(hgv, approved) \wedge concern(hgv, addressed) \quad (4.48)$$

If a design raised no concerns, or the concerns have been addressed, the Channel Tunnel Safety Authority may approve the design. Once the design has been approved by the Channel Tunnel Safety Authority, the Intergovernmental Commission may grant the permit for the design:

$$status(D, approved) \rightarrow P(CT, Approve\_Safety(D)) \quad (4.49)$$

$$[CT, Approve\_Safety(D)]ctsaApproval(D) \quad (4.50)$$

$$igcSupervise \wedge ctsaApproval(D) \rightarrow P(igc, Grant\_Permit(D)) \quad (4.51)$$

$$[I, Grant\_Permit(D)]permit(D) \quad (4.52)$$

Implementing the Designs

As discussed in Section 4.1, the Eurotunnel management must properly implement the designs to ensure an acceptable level of safety. The assumption is made that the supervision of the Intergovernmental Commission obliges the implementation of the policies in the system designs, as described in Quote 2. As discussed above, the staff should be trained as part of the implementation of the general system design:

$$igcSupervise \wedge permit(gen) \wedge \neg trained(e\_staff) \rightarrow IO(e\_man, Train(e\_staff)) \quad (4.53)$$

As with the general design, the Intergovernmental Commission obliges the Eurotunnel management to implement the policies in the HGV design. Again, there is an arbitrary ordering on the implementation of these policies.

$$igcSupervise \wedge permit(hgv) \wedge \neg(implemented(minSepPolicy)) \rightarrow IO(e\_man, Implement(minSepPolicy)) \quad (4.54)$$

$$igcSupervise \wedge permit(hgv) \wedge (implemented(minSepPolicy)) \wedge \neg(implemented(dgPolicy)) \rightarrow IO(e\_man, Implement(dgPolicy)) \quad (4.55)$$

$$igcSupervise \wedge permit(hgv) \wedge (implemented(dgPolicy)) \wedge \neg(trained(e\_staff)) \rightarrow IO(e\_man, Train(e\_staff)) \quad (4.56)$$

$$[E, Implement(P)]implemented(P) \quad (4.57)$$

From the above rules, the implementation of either design requires the training of the staff:

$$[E, Train(ES)]trained(ES) \tag{4.58}$$

However, the report hints in Paragraph 2/8, and states in Paragraphs 19/8, 30/8, and 15/9, that the staff are insufficiently trained:

> **Quote 10:** The lack of training or experience of Eurotunnel staff in the management of emergencies, was apparent. For some key personnel this was their first experience of a major emergency. (15/9)

This is expressed in the model as follows:

$$true \rightarrow \neg trained(e\_staff) \tag{4.59}$$

The Channel Tunnel system was operating in an observably normative state until the fire, so the model includes an axiom stating that the omission of the staff training action is a latent failure:

$$latento(e\_man, Train(e\_staff)) \tag{4.60}$$

Bi-National Test

Prior to the accident, a test exercise was performed. We assume that such a test exercise is weakly obliged and permitted in connection with the Intergovernmental Commission's supervision and the design permits being granted.

$$\begin{aligned} &igcSupervise \wedge permit(hgv) \wedge permit(gen)) \rightarrow \\ &O(e\_man, Test\_Ex)) \wedge P(e\_man, Test\_Ex) \end{aligned} \tag{4.61}$$

The report gives a description of the results of the test in 2/8:

> **Quote 11:** The alerting procedures that Eurotunnel used at the time of the incident were deficient. The procedures had failed to be fully or properly implemented on a number of occasions before; these had been brought to the attention of Eurotunnel. Alerting difficulties had also occurred at the major Bi-National exercise, eight days earlier.

$$[e\_man,\ Test\_Ex]deficientProcedures \wedge improperImplementation \tag{4.62}$$

The *deficientProcedures* predicate is defined with respect to the safety of the design:

$$deficientProcedures \rightarrow \neg\forall D : DESIGN.(safety(D, both) \vee safety(D, unavailable)) \quad (4.63)$$

As we have stated, the Channel Tunnel system was operating in an observably normative state until the fire, so the model includes axioms stating that the qualitative failure to produce designs of the appropriate standard and the failure of the Channel Tunnel Safety Authority to ensure the safety of the designs are latent failures (Clauses 4.64 and 4.65):

$$latentc(dc, Produce\_Design(D, well\_designed\_reliable\_equipment)) \quad (4.64)$$

$$(concern(D, unaddressed)) \wedge safety(D, well\_designed\_reliable\_equipment) \rightarrow latentc(ctsa, Approve\_Safety(D)) \quad (4.65)$$

4.3 Summary

This section has outlined and axiomatised a subset of the behaviour in the Channel Tunnel fire accident report. Included within these axioms are the regulatory and managerial safety frameworks. In the following chapter, we present three theorems that demonstrate that these frameworks failed to ensure safety in the model.

Chapter 5

Reasoning about an Accident Report

In this chapter, we demonstrate the benefits of reasoning about the entities, properties, and behaviour represented in an EDAL model of the Channel Tunnel fire report.

The 'wide scope' approach to the initial stages of the modelling process and the construction of a formal model led to the formulation of many interesting hypotheses, in addition to the usual hypotheses that validate the report's conclusions. Many inconsistent statements were found in the report, not all of which are due to the problems associated with natural language representation. For example, issues are raised in the body of the report that are not dealt with in the conclusions: given the concerns of the Channel Tunnel Safety Authority, why were the system designs approved; why was there no reaction from the Eurotunnel Management to the poor performance of the Eurotunnel Staff in the training exercises; why were the First Line of Response team given inaccurate information on the incident train's location after its true location was known?

This chapter presents the key steps of three deductive proofs. These demonstrate that the agent representing the designers and constructors of the Channel Tunnel system makes a qualitative failure in producing the designs in the model, that the prescriptive behaviour of the agent representing the Channel Tunnel Safety Authority is insufficient to ensure the safety of the designs, and that the agent representing the Eurotunnel management makes an error of omission in failing to train the Eurotunnel staff. While these theorems examine the roles of entities in the report, they are not necessarily indicative of the actual events. The relationships described in the model are open to other interpretations.

The theorems demonstrate EDAL's ability to highlight prescription conflicts, errors of omission, and qualitative failures. Proving the theorems requires reasoning with the deontic and modal operators of EDAL. The theorems are also interesting because they

affirm some of the findings of the report, and also indicate potential errors elsewhere in the report and in the system (as represented in the report).

5.1 Theorem One: Lack of Staff Training

The first theorem states that the report model demonstrates that the Eurotunnel Management failed to ensure an acceptable level of safety. The initial scenario, from which this is shown, represents an early stage in the history of the Channel Tunnel system, before the Treaty of Canterbury was signed:

$$\begin{aligned} init = \quad & \eta \wedge (\neg signed(cantTreaty)) \wedge \\ & (status(hgv, unfinished) \wedge \\ & (status(gen, unfinished) \wedge \\ & (\neg trained(e\_staff) \end{aligned} \tag{5.1}$$

The theorem is formalised as follows:

$$scen \quad \vdash \quad \exists \sigma : AS(Act).[[\sigma]] \neg emanALoS$$

Given the initial scenario, there is a sequence of actions following which $emanALoS$ is negated. To prove this, we must demonstrate that the Eurotunnel management failed to implement the relevant policies after receiving a permit. We focus on the Eurotunnel management's failure to train the staff adequately.

Space requirements prevent the presentation of the full formal proofs in this chapter. Instead, the proofs are largely presented informally, with crucial portions given in more detail and the current scenario given at key points. Properties that hold in the scenario, but play no further part in the theorem, are omitted. Unless otherwise stated, the steps use the EDAL proof rules (rules 2.1, 2.2, 2.3, 2.4) and the Necessitation rule (rule 7 in Section 2.3.4). Unless otherwise stated, performed actions are permitted or obliged by axioms in the model.

Generating the Designs

In the initial scenario $init$, the designs of the HGV and general train systems are not complete: $(status(hgv, unfinished) \wedge (status(gen, unfinished))$. Assuming that the designers and constructors produce the designs (in an arbitrary order), we can apply axioms 4.4, 4.8 and 4.9 to deduce the following scenario:

$$[dc, Produce\_Design(gen, S)][dc, Produce\_Design(hgv, S)]$$
$$status(hgv, unexamined) \wedge$$
$$status(gen, unexamined) \wedge$$
$$\neg signed(cantTreaty) \wedge \neg trained(e\_staff) \wedge \eta \quad (5.2)$$

Recall that the normative constant η holds in each scenario reached by performing prescribed actions.

Signing the Treaty

Signing the Treaty of Canterbury is permitted (4.10) and obliges the establishment of the Channel Tunnel Safety Authority (4.13) and the Intergovernmental Commission (4.14). The establishment of the two bodies asserts the predicates $established(ctsa)$ (4.15) and $established(igc)$ (4.16). Following the performance of these actions, the following scenario can be deduced:

$$[dc, Produce\_Design(gen, S)][dc, Produce\_Design(hgv, S)]$$
$$[ukf, Sign(cantTreaty)][gm, Establish(ctsa))][gm, Establish(igc)]$$
$$established(igc) \wedge established(ctsa) \wedge$$
$$igcSupervise \wedge safetyAdvise(ctsa, igc) \wedge$$
$$status(hgv, unexamined) \wedge$$
$$status(gen, unexamined) \wedge$$
$$\neg trained(e\_staff) \wedge \eta \quad (5.3)$$

Applying for Permits

Once the designs are finished, Eurotunnel management must send them to the Intergovernmental Commission for scrutiny before a permit is granted (4.18, 4.19). The Intergovernmental Commission sends the system designs to the Channel Tunnel Safety Authority for safety assessment (4.20, 4.21). In this model of behaviour, we assume the general permit is applied for first, although the order is unimportant.

Assessing the Designs

The Channel Tunnel Safety Authority arranges for designs sent for safety assessment to be examined by technical experts (4.23, 4.24). After the general design has been sent from the Eurotunnel management to the Intergovernmental Commission to the Channel Tunnel Safety Authority, the following scenario can be deduced:

$$
\begin{aligned}
&[dc, Produce\_Design(gen, S)][dc, Produce\_Design(hgv, S)]\\
&\quad [ukf, Sign(cantTreaty)][gm, Establish(ctsa))][gm, Establish(igc)]\\
&\quad [e\_man, Apply\_Permit(gen)][igc, Safety\_Assess(gen)]\\
&\quad [ctsa, Arrange\_Scrutiny(gen)]\\
&\qquad igcSupervise \wedge arrangedScrutiny(gen) \wedge\\
&\qquad safetyAdvise(ctsa, igc) \wedge status(hgv, unexamined) \wedge\\
&\qquad status(gen, unexamined) \wedge\\
&\qquad \neg trained(e\_staff) \wedge \eta
\end{aligned} \tag{5.4}
$$

Assessment Results

As described in the report, the general designs pass scrutiny (4.25, 4.26). Having been assessed, and any concerns having been addressed, a design is approved by the Channel Tunnel Safety Authority (4.49, 4.50). Once the design has been approved, the Intergovernmental Commission can issue a permit (4.51, 4.52).

As described in the report, the general service permit is granted. This results in the following scenario:

$$
\begin{aligned}
&[dc, Produce\_Design(gen, S)][dc, Produce\_Design(hgv, S)]\\
&[ukf, Sign(cantTreaty)][gm, Establish(ctsa)][gm, Establish(igc)]\\
&[e\_man, Apply\_Permit(gen)]\\
&[igc, Safety\_Assess(gen)]\\
&[ctsa, Arrange\_Scrutiny(gen)]\\
&[tech\_ex, Examine\_Safety(gen)]\\
&[ctsa, Approve\_Safety(gen)]\\
&[igc, Grant\_Permit(gen)]\\
&\quad permit(gen) \wedge igcSupervise \wedge status(gen, approved) \wedge\\
&\quad status(hgv, unexamined) \wedge concern(gen, none) \wedge\\
&\quad ctsaApproval(gen) \wedge \neg trained(e\_staff) \wedge \eta
\end{aligned} \tag{5.5}
$$

General Service Implementation

The supervision of the Intergovernmental Commission obliges the implementation of the policies in the general system design (4.53). However, the report states that the staff are insufficiently trained (4.59).

From this we can deduce $\exists \sigma : AS(Act).[[\sigma]] \neg emanALoS$, as follows:

1.	Axiom 4.3	$emanALoS \rightarrow$ $(permit(hgv) \rightarrow$ $\forall \alpha, \beta, \gamma : Act.[e\_man, \alpha][e\_man, \beta][e\_man, \gamma]$ $(trained(e\_staff) \wedge implemented(minSepValue) \wedge$ $implemented(dgPolicy))$ $\wedge (permit(gen) \rightarrow$ $\forall \alpha : Act.[e\_man, \alpha] trained(e\_staff)))$
2.	Weakening (1)	$emanALoS \rightarrow$ $(permit(gen) \rightarrow$ $\forall \alpha : Act.[e\_man, \alpha] trained(e\_staff))$
3.	Contrapos. (2)	$\neg(permit(gen) \rightarrow$ $\forall \alpha : Act.[e\_man, \alpha](trained(e\_staff)))$ $\rightarrow \neg emanALoS$
4.	Defn. of $\rightarrow$ (3)	$\neg(\neg permit(gen)) \vee \forall \alpha : Act.[e\_man, \alpha] trained(e\_staff))$ $\rightarrow \neg emanALoS$
5.	de Morg. (4)	$permit(gen) \wedge \neg \forall \alpha : Act.[e\_man, \alpha] trained(e\_staff))$ $\rightarrow \neg emanALoS$
6.	Axiom 4.59	$true \rightarrow \neg trained(e\_staff))$
7.	Necess. (6)	$\forall \alpha : Act.[e\_man, \alpha] \neg trained(e\_staff))$
8.	EDAL Rule E8	$\forall \alpha : Act. \neg [e\_man, \alpha] trained(e\_staff)$
9.	Pred. Logic (8)	$\neg \forall \alpha : Act.[e\_man, \alpha] trained(e\_staff)$
10.	Scenario 5.5	$permit(gen)$
11.	Mod. Pons. (5,9,10)	$\neg emanALoS$

This accords with the view of the report author, who makes the following recommendation, based on Paragraph 19/8:

> **Recommendation 17:** Eurotunnel must improve the training of all of their staff in relation to the management of emergencies and develop a structured and practically based training programme.

The failure to adequately train the staff by the Eurotunnel management does not affect the observable behaviour of the system, at this stage (4.60). The latent failure operators enable the action to be omitted and the system to remain in an observably normative scenario (provided the action performed instead is permitted or immediately obliged).

5.2 Theorem Two: Deficient Procedures in Designs

The second theorem states that the report model demonstrates that the designers and constructors failed to ensure an acceptable level of safety. This theorem is expressed

as follows:

$$init \vdash \exists\sigma : AS(Act).[[\sigma]]\neg dcALoS$$

Given the initial scenario, there is a sequence of actions following which $dcALoS$ is negated. To prove this, we must demonstrate that the designs produced by the designers and constructors did not incorporate both well designed, reliable equipment and clear, effective procedures. We focus on the fact that the procedures of the designs were found to be deficient. This proof builds on the steps of the first proof.

Application and Scrutiny of HGV Design

After the general permit is granted, the next action the Eurotunnel management performs is the application for the HGV Service Permit (4.19). The same permit application actions are performed in the application for a permit for the HGV design, leading to the following scenario:

$$\begin{aligned}
&[dc, Produce_Design(gen, S)][dc, Produce_Design(hgv, S)]\\
&\quad [ukf, Sign(cantTreaty)][gm, Establish(ctsa)][gm, Establish(igc)]\\
&\quad [e_man, Apply_Permit(gen)]\\
&\quad [igc, Safety_Assess(gen)]\\
&\quad [ctsa, Arrange_Scrutiny(gen)]\\
&\quad [tech_ex, Examine_Safety(gen)]\\
&\quad [ctsa, Approve_Safety(gen)]\\
&\quad [igc, Grant_Permit(gen)]\\
&\quad [e_man, Apply_Permit(hgv)]\\
&\quad [igc, Safety_Assess(hgv)]\\
&\quad [ctsa, Arrange_Scrutiny(hgv)]\\
&\quad [tech_ex, Examine_Safety(hgv)]\\
&\quad\quad igcSupervise \wedge ctsaApproval(gen) \wedge concern(gen, none) \wedge\\
&\quad\quad status(hgv, unapproved) \wedge status(gen, approved) \wedge\\
&\quad\quad permit(gen) \wedge \eta
\end{aligned} \tag{5.6}$$

However, the technical experts express concern about the design (4.27). The expression of these concerns by the Channel Tunnel Safety Authority (4.29) and the reaction by the Eurotunnel management (4.31 to 4.48) leads to the following scenario:

$$\begin{aligned}
&[dc, Produce_Design(gen, S)][dc, Produce_Design(hgv, S)]\\
&\quad [ukf, Sign(cantTreaty)][gm, Establish(ctsa)][gm, Establish(igc)]
\end{aligned}$$

$[e\_man, Apply\_Permit(gen)]$
$[igc, Safety\_Assess(gen)]$
$[ctsa, Arrange\_Scrutiny(gen)]$
$[tech\_ex, Examine\_Safety(gen)]$
$[ctsa, Approve\_Safety(gen)]$
$[igc, Grant\_Permit(gen)]$
$[e\_man, Apply\_Permit(hgv)]$
$[igc, Safety\_Assess(hgv)]$
$[ctsa, Arrange\_Scrutiny(hgv)]$
$[tech\_ex, Examine\_Safety(hgv)]$
$[ctsa, Express\_Concerns(hgv)]$
$[e\_man, Request(cfds, Reaction\_Test)]$
$[cfds, Reaction\_Test]$
$[e\_man, Request(rtms, Separation\_Test)]$
$[rtms, Separation\_Test]$
$[e\_man, Request(tester, HGV\_RoF\_Dev\_Test)]$
$[tester, HGV\_RoF\_Dev\_Test]$
$[e\_man, Request(tester, Em\_Proc\_Test)]$
$[tester, Em\_Proc\_Test]$
$[e\_man, Request(ems, Vent\_Test)]$
$[ems, Vent\_Test]$
$[e\_man, PresentTestEvidence]$

$$igcSupervise \wedge ctsaApproval(gen) \wedge concern(gen, none) \wedge status(hgv, approved) \wedge status(gen, approved) \wedge concern(hgv, addressed) \wedge permit(gen) \wedge \eta \quad (5.7)$$

The concerns having been addressed, the HGV design is approved by the Channel Tunnel Safety Authority and the HGV permit granted as with the General design.

HGV Service Implementation

The Intergovernmental Commission obliges the Eurotunnel management to implement the policies in the HGV design (4.54 to 4.58).

Bi-National Test

The test exercise is weakly obliged and permitted as part of the HGV and general design permits being granted (4.61). After the performance of the test exercise, the following scenario holds:

$$
\begin{array}{l}
[dc, Produce\_Design(gen, S)][dc, Produce\_Design(hgv, S)] \\
\quad [ukf, Sign(cantTreaty)] \\
\quad [gm, Establish(ctsa)][gm, Establish(igc)] \\
\quad [e\_man, Apply\_Permit(gen)] \\
\quad [igc, Safety\_Assess(gen)] \\
[ctsa, Arrange\_Scrutiny(gen)] \\
[tech\_ex, Examine\_Safety(gen)] \\
[ctsa, Approve\_Safety(gen)] \\
[igc, Grant\_Permit(gen)] \\
[e\_man, Apply\_Permit(hgv)] \\
[igc, Safety\_Assess(hgv)] \\
[ctsa, Arrange\_Scrutiny(hgv)] \\
[tech\_ex, Examine\_Safety(hgv)] \\
[ctsa, Express\_Concerns(hgv)] \\
[e\_man, Request(cfds, Reaction\_Test)] \\
[cfds, Reaction\_Test] \\
[e\_man, Request(rtms, Separation\_Test)] \\
[rtms, Separation\_Test] \\
[e\_man, Request(tester, HGV\_RoF\_Dev\_Test)] \\
[tester, HGV\_RoF\_Dev\_Test] \\
[e\_man, Request(tester, Em\_Proc\_Test)] \\
[tester, Em\_Proc\_Test] \\
[e\_man, Request(ems, Vent\_Test)] \\
[ems, Vent\_Test] \\
[e\_man, Present\_Evidence] \\
[ctsa, Approve\_Safety(hgv)] \\
[igc, Grant\_Permit(hgv)] \\
[e\_man, Implement(minSepPolicy)] \\
[e\_man, Implement(dgPolicy)] \\
[e\_man, Test\_Ex] \\
\quad deficientProcedures \wedge \\
\quad concern(gen, none) \wedge concern(hgv, addressed) \wedge \\
\quad status(gen, approved) \wedge status(hgv, approved) \wedge \\
\quad ctsaApproval(gen) \wedge ctsaApproval(hgv) \wedge \eta
\end{array}
\tag{5.8}
$$

From this, we can conclude $\exists\sigma : AS(Act).[[\sigma]]\neg dcALoS$:

1. Axiom 4.1 $dcALoS \rightarrow$

		$\forall D : DESIGN.safety(D, both) \vee safety(D, unavailable)$
2.	Contrap. (1)	$\neg(\forall D : DESIGN.safety(D, both) \vee safety(D, unavailable)) \rightarrow \neg dcALoS$
3.	Axiom 4.63	$deficientProcedures \rightarrow \neg\forall D : DESIGN.(safety(D, both) \vee safety(D, unavailable))$
4.	Scenario 5.8	$deficientProcedures$
5.	Mod. Pons. (3,4)	$\neg\forall D : DESIGN.(safety(D, both) \vee safety(D, unavailable))$
6.	Mod. Pons. (2,5)	$\neg dcALoS$

Although this theorem is not explicit in the report, we have demonstrated that it can be deduced from the argumentation in the report.

5.3 Theorem Three: Inadequate Design Scrutiny

The third theorem states that the report model demonstrates that the Channel Tunnel Safety Authority failed to ensure an acceptable level of safety. This theorem is expressed as follows:

$$init \vdash \exists \sigma : AS(Act).[[\sigma]]\neg ctsaALoS$$

Given the initial scenario, there is a sequence of actions following which $ctsaALoS$ is negated. As the definition of *ctsaAlos* states, one duty of the Channel Tunnel Safety Authority is assumed to be the detection of poor designs. As well as requiring approved designs to be free from concerns or to have had concerns addressed, $ctsALoS$ also depends on the design having both clear, effective procedures and well designed, reliable equipment. To prove this theorem, we demonstrate that the designs used in the Eurotunnel system did not have both properties.

The proof of this theorem builds on the steps of the second proof, as follows:

1.	Axiom 4.2	$ctsaALoS \rightarrow \forall D : DESIGN.ctsaApproval(D) \rightarrow (\neg concern(D, unaddressed) \wedge safety(D, both))$
2.	Scenario 5.8	$\forall D : DESIGN.ctsaApproval(D)$
3.	Mod. Pons. (1,2)	$ctsaALoS \rightarrow \forall D : DESIGN. (\neg concern(D, unaddressed) \wedge safety(D, both))$
4.	Scenario 5.8	$\forall D : DESIGN.\neg concern(D, unaddressed)$
5.	Conjunction (3,4)	$ctsaALoS \rightarrow \forall D : DESIGN.safety(D, both))$

6.	Contrap. (5)	$\neg\forall D : DESIGN.safety(D, both) \rightarrow \neg ctsaALoS$
7.	Proof 2, Step 5	$\neg\forall D : DESIGN.$ $\neg(safety(D, both) \wedge \neg safety(D, unavailable)$
8.	defn. of $\exists$ (7)	$\exists D : DESIGN.$ $\neg(safety(D, both) \wedge \neg safety(D, unavailable)$
9.	Scenario 5.8	$status(gen, approved)$
10.	Scenario 5.8	$status(hgv, approved)$
11.	Axiom 4.8	$status(D, DS) \wedge DS \neq unfinished \rightarrow$ $safety(D, S) \wedge S \neq unavailable$
12.	Mod. Pons (9,11)	$safety(gen, S) \wedge S \neq unavailable$
13.	Mod. Pons (10,11)	$safety(hgv, S) \wedge S \neq unavailable$
14.	Induction (12,13)	$\forall D : DESIGN.\neg safety(D, unavailable)$
15.	Disjunc. (8,14)	$\exists D : DESIGN.\neg(safety(D, both)$
16.	Defn. of $\exists$ (15)	$\neg\forall D : DESIGN.safety(D, both)$
17.	Mod. Pons. (6,16)	$\neg ctsaALoS$

Based on the assumptions in the model and the behaviour described in the report, the prescripted behaviour of the Channel Tunnel Safety Authority permits unsafe designs. The technical experts should have been consulted again, once the Eurotunnel management presented their evidence regarding the safety of the HGV design, to ensure the safety of the proposed system.

We have shown that, in this model, three agents break their duty to ensure an acceptable level of safety. The designers and constructors produce designs with deficient procedures, the Channel Tunnel Safety authority accept these designs, and the Eurotunnel management fail to train their staff. However, of these bodies the report criticises only the Eurotunnel management.

We have shown that these duties were all broken prior to any of the events from the fire.

5.4 Discussion

This section discusses some of the problems and issues encountered during the reasoning process.

5.4.1 EDAL Issues

The main purpose of the work presented in this chapter is to demonstrate that the deontic and action operators can be used to reason about accident reports. As demonstrated, the action operators facilitate the study of the actual behaviour as the scenarios

changed. The deontic operators made it possible to describe and analyse where, how, and by whom, norms are broken within the system. They also enabled us to model situations where the prescriptive behaviour is inadequate, with respect to preventing the violation of a system constraint. Perhaps most valuably, the deontic operators enabled us to focus on aspects of the regulatory and managerial safety frameworks in the system. Recent accident analysis literature focuses on organisational accidents and highlights the importance of safety management and regulation issues (i.e., Reason [Rea97]), and yet the focus on most work in formal accident analysis is still on modelling observable operator error [Joh99].

The theorems presented in this chapter may seem straightforward to the reader. However, it should be noted that the information required for the proofs is scattered throughout the accident report. It is only through the production of the model that this information is collected together. The proofs further increase the readers confidence in the theorems. The theorems illustrate how reasoning with EDAL can reveal properties of a system that are not immediately apparent from the report. This is a valuable addition to the ability to validate the report conclusions. The theorems were chosen because they demonstrate that EDAL can be used to reason about qualitative failures, errors of commission and omission, and prescriptive failings.

5.4.2 Normativity Issues

The latent failure operators, *latento* and *latentc*, enable EDAL to model observable normativity. However, even with these operators, the binary nature of normativity is a restricting factor in the analysis of accidents.

Our dissatisfaction with the current representation of normativity in EDAL led us to explore other, more sophisticated definitions of normativity, of which we will present four.

The first approach we considered uses two normative constants, one for observable normativity and the other for absolute normativity. The intention was to provide a system in which scenarios with latent conditions are modelled as non-normative (by negating the absolute normativity constant), but non-catastrophic (as observable normativity is still present). However, this approach gives little improvement on the existing approach, as the constant representing absolute normativity is negated early in any realistic model.

The second approach features multiple values, or *levels*, of non-normativity, rather than the binary value of EDAL. This makes it possible to distinguish between the severity of different non-normative scenarios. For example, a scenario in which there is a fire and the staff are poorly trained is a more serious non-normative scenario than one in which the staff are poorly trained. However, defining levels of severity proved problematic. Every action performed in a non-normative scenario is deemed non-normative, yet performing a non-normative action does not necessarily increase the severity of the scenario. For example, the action of extinguishing the fire in the more severe of the two scenarios described would reduce the severity of that scenario without the scenario

becoming normative. Actions could be weighted by their severity to make their effect on the level more realistic. This would require large amounts of domain specific information, particularly as the severity of an action may be largely scenario dependent. The information given in an accident report is insufficient for such an approach to be feasible.

The third approach involves the localisation of normativity and is based on the work of Herrestad and Krogh [HK95]. This approach allocates a normative constant to each agent. If an agent performs a non-normative action, only the agent and the components of the system directly affected by the non-normative action are affected. The behaviour exhibited by other components that do not interact with this component remains unaffected. For example, if the staff are poorly trained in emergency procedures, this fact does not affect the passengers or the emergency services unless they are forced to interact with the staff in an emergency situation. This work has yet to be applied to anything on the scale of the Channel Tunnel fire report.

Finally, we looked at the use of defeasible reasoning methods with deontic logics. In defeasible deontic logics, norms are ranked by importance, relevance, or priority, using a preference relation. The introduction of a contradictory norm to the scenario requires the removal of at least one norm with a lower ranking, such that a consistent set remains [MW93]. Defeasible reasoning is generally used to model inference with incomplete knowledge, but norm prioritisation can also be used to represent the strategies used for resolving conflicting goals. One of the best examples of coping with conflicting norms comes from the North Anna incident, described in Appendix B.3. The operators' used a decision process, akin to a preference relation, to decide between following the procedures of the controlling body and following their intuition, which resulted from experience with nuclear reactors. Expert advice would be required to determine the highly subjective judgements that prioritise the various norms. Although defeasible reasoning could potentially be very useful in safety engineering, it requires too much information about the system and the agents to be useful in modelling existing accident reports.

5.4.3 Interpreting the Results

It is worth reiterating at this point that the theorems presented do not 'prove' anything about the *actual* behaviour of the various bodies involved. Indeed, given that the source, the accident report, is informal, the model cannot be entirely faithful.

The exercise *has* increased our confidence that the report can be interpreted as stating that the Channel Tunnel Safety Authority, the designers and constructors, and Eurotunnel management all failed in their obligation to ensure an acceptable level of safety.

A strong justification for this modelling approach is that it draws out implicit information. This is useful to report writers, who need to know if the report implies information that is incorrect, that should have been explicit, or that should have been omitted. It is useful to system designers, who may need to know details about the system that the report does not make explicit.

Errors are common in mathematical proofs, and increases in the size and complexity of the proof increase their propensity [Kne97, Har96]. Automating reasoning in EDAL would reduce the likelihood of errors being introduced and increase our confidence in the reasoning process. This is addressed in the following chapter, in which we present an interpreter for EDAL.

There are strong arguments mandating the search for a more usable, less formal, means of reasoning about the model. The semiformal steps of SCS assist in the production of an unambiguous and consistent model of the report. However, a large amount of additional work is required to formally reason about the model that these steps produce. As demonstrated, the semiformal processes of SCS uncover many errors and omissions without the need for any formal reasoning. This can be attributed to the distinct focus of each step and the novel views of the system that these steps create. Furthermore, the analyst's familiarity with the model and the system is also an extremely valuable asset in the discovery of errors. These findings reflect those of other users of requirements specification techniques [MLR+97]. In accident report modelling, the confidence imbued by formal reasoning is limited considerably by the number of assumptions and the layers of interpretation present in both the report and the model.

A more 'lightweight' approach could be taken to the formalisation process. Easterbook et al. [ELC+98] report that such an approach improves consistency and confidence in the products of the semiformal techniques, and highlights errors that may be undetected by inspection and traceability analysis. The formal model could be used to generate a representation of the system in another language that better supports animation or visualisation of the behaviour. This could be a programming language, or an executable state-based language, such as state charts [Har87]. Techniques, such as State Machine Hazard Analysis, could then be used to explore the potential for further hazardous states [Lev95]. Furthermore, these state-based systems have been shown to be scalable and accessible to non-logicians. State charts, generated from formal specifications, can improve the visualisation of the specification for both logicians and non-logicians [Gur95]. In addition, the presentation facilitates the detection of errors and are supported by automated animation tools (i.e., STATEMATE [HLN+90]).

Further work is required to evaluate which approach would return the more valuable information. What is certain is that manual reasoning is a time-consuming process and the conclusions are based on the large number of assumptions that are required to model an accident report.

5.5 Summary

This chapter demonstrates that EDAL can be used to reason about qualitative failures, errors of omission and commission, and prescriptive failings. It has demonstrated that the deontic operators are useful for modelling the managerial and regulatory frameworks that must be examined in organisational accidents. The size and complexity of the proofs of the three theorems presented illustrate that the manual proof process for

EDAL has the potential for the introduction of errors. In the following chapter, we present an interpreter that reduces this potential, and improves our ability to examine the formal model and to evaluate EDAL.

Chapter 6

An Executable Deontic Action Language

6.1 Introduction

Systems and accidents are increasing in complexity [Lev95]. The previous chapters demonstrated the complexity of formal modelling and reasoning. An EDAL interpreter tool would facilitate more reliable reasoning, more accurate and accessible specifications [Hal90], and rapid prototyping of the language and model [KM87, Fuc92]. In this chapter, we present a Prolog based interpreter for an executable version of EDAL.

Prolog and EDAL are quite dissimilar. In particular, Prolog has no notion of actions or states. In this chapter, we discuss the interpreter, highlighting in particular the interesting problems that are faced when modelling an action-based language in Prolog's horn-clause logic.

6.2 Preliminaries

In this section, we describe Prolog and the executable subset of EDAL, which we refer to as DALEX.

6.2.1 Prolog

Prolog is an executable first-order predicate calculus. Where conventional programming languages follow sequences of instructions, in Prolog, a 'program' is a set of facts and a set of rules that are used to solve a goal.

In Prolog, variables are unsorted. The first letter of all variable names is in upper case, predicates and atoms begin with a lower case letter. In this chapter, Prolog code is given in teletype.

Facts are statements of object properties `x.` or relations between objects `r(x,y)..` Horn clauses, of the form `_:-_,` express rules in Prolog. For example, the following rule states that if `y1` and `y2` hold, `x` holds: `x :- y1, y2..` Collections of rules with the same predicate in the head of the horn clause (i.e., on the left of the horn clause) are referred to as procedures.

6.2.2 DALEX

The development of executable modal logics is a research area in its own right. This chapter, therefore, represents an initial feasibility study into the application of such systems for accident report analysis. The executable language, DALEX, is a subset of EDAL. One notable difference is that the action description operator does not currently have an explicit agent parameter. The EDAL operators featured in DALEX are immediate permission `ip`, immediate obligation `io`, weak permission `p`, and weak obligation `o`. The normative constant in DALEX is `norm`.

6.3 Implementation Issues

This section discusses the differences between Prolog and EDAL that increase the complexity of producing a Prolog interpreter for EDAL.

6.3.1 Negation as Failure

The Closed World Assumption, as used in EDAL, states that if a property P is not derivable, then $\neg P$ is derivable (see clause 2.4). The DALEX interpreter, and Prolog in general, only partially models first-order negation because the horn clause cannot be used to derive negative properties [Cla78]. Instead the less powerful Negation as Failure rule is used. This rule states that a negative property `not P` is derivable if it is provable that `P` cannot be proved.

With a closed world assumption this is generally not problematic, as negative properties can be deduced by their absence. For example, if the interpreter states that, in the current scenario:

```
moving(train1, west),
p(stop(driver1,train1))
```

Many other properties can be deduced from this, such as:

```
not(moving(train1, east),
not(not(p(stop(driver1,train1)))),
not(p(stop(driver2,train1)))
```

DALEX requires the `neg` operator to enable explicit assertion of negative properties and to express negative conditions in a conditional. This operator is used to denote explicit negation of a property in a formula. For example, the action description `modal(X, neg(p(stop(train(t1)))))` states that following the performance of action `X`, the property `p(stop(train(t1)))` is negated. Once the action has been performed, the interpreter removes the property from the scenario database.

6.3.2 Accessibility Relation

An interpreter of any modal language must model the language's accessibility relation. The *accessibility relation* is a relation that, for a given scenario s_i, enumerates the scenarios that are accessible from s_i. In an action-based languages, such as DALEX, a scenario s_{i+1} is *accessible* from scenario s_i if the set of properties that are known to hold in s_{i+1} are the set of properties known to hold after an agent has performed an action in s_i. For example, property p is known to hold in the scenario following the performance of an action a iff the modal formula $[a]p$ is known to hold in the current scenario (see figure 6.1).

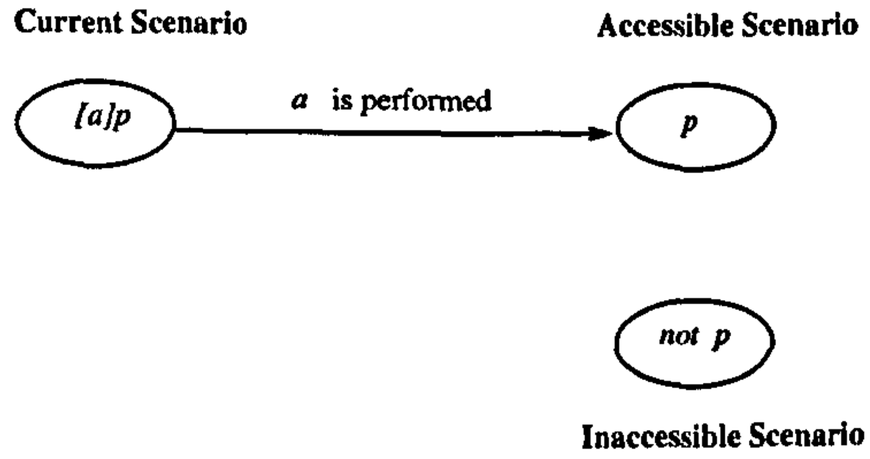

Figure 6.1: Accessibility of Scenarios in DALEX.

In EDAL, the next scenario is accessed only when an action is performed. The current DALEX interpreter models the action-based language accessibility relation using a trace of actions and a recursive procedure:

```
behaviour(_,_,[]).
behaviour(..., [H|T]):-
        ...
        behaviour(..., T).
```

The list `[H|T]` represents the trace of actions to be performed. The first action in the list `H` is viewed as the action performed to reach the current scenario. The recursive

call to the `behaviour` procedure passes the tail of the list of actions into the next 'scenario'. We discuss how the properties that hold in each scenario are calculated later.

6.3.3 Managing the Database

The DALEX interpreter models the EDAL scenario theory as a list of properties, which we call the *scenario database*. Where the EDAL semantic axioms define manipulation of a theory, the DALEX axioms define manipulation of the scenario database. The dynamic nature of the scenario database makes it possible for conflicts to be introduced into the database. Such conflicting information prevents valid inference. Consider the following scenario, in which a train is travelling east:

```
[moving(train1, east),
 modal(stop(train1), moving(train1, null))]
```

If the `stop(train1)` action is performed, `moving(train1, null)` will hold. However, without an explicit conflict avoidance strategy, `moving (train1, east)` will also hold. Properties persist in the database, so a conflict potentially arises each time a property is changed in a procedural interpreter. However, not all effects introduce conflicts. For example, more than one passenger can be aboard a single train without a conflict: `aboard(p1,t1), aboard(p2,t1).`

6.3.4 Translating Between Languages

In this section, we discuss the process of translating the semantics of EDAL into the clausal form of Prolog. As the interpreter is used to validate the EDAL semantics and to model EDAL behaviour, there should be a formal link between the EDAL operators and their DALEX counterparts.

Translating the Deontic Operators

The translation begins with the removal of explicit agents from the EDAL operators. We then use the translation process from predicate calculus to clausal form, given in Clocksin and Mellish [CM87]. This involves:

1 Removal of implication,

2 Moving negation inwards,

3 Skolemisation,

4 Moving universal quantifiers outwards,

5 Distributing $\vee$ over $\wedge$,

6 Putting the results into clauses.

The predicate calculus transformation process can be applied to the translation of the deontic operators because the semantics of the deontic operators are defined without *explicit* reference to future theories. Modal formulae are used instead. For example, consider the (agentless) model theory definition of EDAL rule E13:

$$\forall \alpha.\eta \in TH(s) \wedge IP(\alpha) \in TH(s) \rightarrow [\alpha]\eta \in TH(s)$$

By using a modal formula, this implication asserts that η holds in the theory following α without explicitly referencing any other theories than the current one.

The translation process for this rule is as follows, with the numbers corresponding to the steps given above:

1. $\forall \alpha.\neg(\eta \wedge IP(\alpha)) \vee [\alpha]\eta$
2. $\forall \alpha.\neg\eta \vee \neg IP(\alpha) \vee [\alpha]\eta$
3. Not Applicable
4. $\neg\eta \vee \neg IP(\alpha) \vee [\alpha]\eta$
5. Not Applicable
6. $[\alpha]\eta :- IP(\alpha), \eta$

The version of this rule in the interpreter is necessarily slightly different, due to the naming conventions of variables, functions and constants in Prolog and the way the modal operator and the normative constant are defined. In addition, the membership of a theory is modelled in DALEX by the membership of the scenario database:

```
member(modal(A,norm),DB) :- member(ip(A),DB),member(norm,DB)
```

As the declarative interpreter uses a separate database from Prolog, the above rule has to be adapted to enable the implication to apply to the scenario database. The procedure that does this is named `addPrescriptions` and is described below.

The translation process increased our understanding of the operators and their interdependencies. During the development of EDAL, translating these semantics revealed an underlying similarity between all the operators, except immediate permission. Additionally, some EDAL semantics were shown to be unnecessary, once converted to clausal form. For example, the DAL rule concerning the relationship between normativity and permission (rule 13) contains an unnecessary double implication, which also conflicts with the latent error operators of EDAL.

However, the latter stages of the translation are not formally defined. Thus, the axioms of DALEX currently have no formal link to EDAL.

Translating the Action Description Operator

The rule defining the action description operator does refer to other theories, therefore we cannot simply abstract away the current theory as we did for the other operators.

The EDAL action description operator rule states: if a property ϕ is known to hold as a consequence of performing an action α in $TH(s)$, then following the performance of the action α, ϕ holds:

$$\forall s \in S, \forall \alpha \in Act.$$
$$if\ [\alpha]\phi \in TH(s)$$
$$then\ \phi \in TH(f_\alpha(s))$$

The `behaviour` procedure given earlier models the movement between actions. A separate procedure `getEffects` is required to establish the properties that follow from the performance of an action.

6.3.5 Meta Rules

The interpreter models the Necessitation rule in a straightforward way. The `behaviour` procedure takes two list parameters representing the properties of the specification (the *specification database*) and the properties of the current scenario (the *scenario database*). By storing the specification database separately, the interpreter avoids the difficulties DAL faced with differentiating scenarios and specifications.

6.4 The DALEX Interpreter

In the first part of this section we describe the current implementation of the DALEX interpreter. In particular, we highlight those rules and procedures that stem from the issues discussed in the previous section. In the second part, we examine the DALEX interpreter's output, giving a trace of a small example.

6.4.1 Interpreter Description

A complete listing of the DALEX interpreter is given in Appendix G. It should be noted that the DALEX interpreter presented does not prove properties of the behaviour. Instead, it executes an action trace and outputs the properties that hold in each scenario. Extending the interpreter to check the properties of the scenarios is straightforward, but the current use of the interpreter is to prototype the operator semantics.

behaviour Procedure

```
behaviour(_,_,[]).
behaviour(SP,DB,[H|T]):-
          getEffects(H,DB,DB,DB1),
          checkNorm(DB1),
          sortConflicts(H,DB1,DB1,DB2),
          removeNegs(SP,DB2,DB2,DB3),
          removeOldNorms(DB3,DB4),
          addPermissions(DB4,DB5),
          addPrescriptions(DB5,DB6),
          behaviour(SP,DB6,T).
```

This is the main procedure of the interpreter. Its parameters are the specification database SP, the scenario database DB, and the trace of actions to be performed [H|T]. When there are no further actions to perform, the terminating condition succeeds.

The effects on the scenario database of performing the action H are identified by a call to getEffects. checkNorm is then called to report if the new scenario is non-normative. Any conflicts created by the effects of performing H are resolved by sort-Conflicts. Any explicit negative properties are then removed by the removeNegs procedure. The new P-structure is then calculated by addPermissions. The prescriptions from the previous scenario are removed by removeOldNorms. The deontically acceptable and unacceptable actions are defined in addPrescriptions. Finally, the execution proceeds when behaviour is recursively called to model the next action. The new database DB6 is passed into the next 'scenario', along with the specification and the remainder of the action trace.

getEffects Procedure

```
getEffects(A,B,C,E):-
        getModEffects(A,B,C,D),
        getFOLEffects(A,D,D,E).
```

This procedure uses the procedures getModEffects and getFOLEffects to determine the new properties that hold in the current scenario. This procedure is based on the proof theory axiom 2.4.

getModEffects Procedure

```
getModEffects(_,DB,[],[]).

getModEffects(A,DB,[modal(A,(modal(A,Z)))|T],
        [modal(A,Z),modal(A,modal(A,Z))|T2]):-
```

```
    \+member(modal(A,Z),DB),
    getModEffects(A,DB,T,T2).
getModEffects(A,DB,[modal(A,(modal(A,Z)))|T],
        [modal(A,modal(A,Z))|T2]):-
    member(modal(A,Z),DB),
    getModEffects(A,DB,T,T2).

getModEffects(A,DB,[modal(A,Y)|T],[modal(A,Y)|T2]):-
    \+(Y=modal(A,Z)),
    \+member(Y,DB),
    getModEffects(A,[Y|DB],[Y|T],T2).
    %Effects of performing the action.
getModEffects(A,DB,[modal(A,Y)|T],[modal(A,Y)|T2]):-
    \+(Y=modal(A,Z)),
    member(Y,DB),
    getModEffects(A,DB,T,T2). %Effects already present

getModEffects(A,DB,[H|T],[H|T2]):-
    \+H=modal(A,Y),
    getModEffects(A,DB,T,T2).
```

`getModEffects` determines the effects of the most recent action, using the modal connectives in the scenario database. These effects are added to the scenario database, assuming that they do not already hold. The parameters of `getModEffects` are (in order): the action performed to reach the current scenario, the database of the previous scenario, the list of elements in this previous database still to be examined by this procedure, and the elements of the new database established by this procedure.

`getFOLEffects` Procedure

```
getFOLEffects(_,D,[],[]).

getFOLEffects(A,D,[X=>Y|T],[X=>Y|T2]):-
    member2(X,D),
    member2(Y,D),
    getFOLEffects(A,D,T,T2).
getFOLEffects(A,D,[X=>Y|T],[X=>Y|T2]):-
    member2(X,D),
    \+ member2(Y,D),
    getFOLEffects(A,[Y|D],[Y|T],T2).
getFOLEffects(A,D,[X=>Y|T],[X=>Y|T2]):-
    \+ member2(X,D),
    getFOLEffects(A,D,T,T2).

getFOLEffects(A,D,[X&Y|T],T2):-
```

```
    member2(X,D),
    member2(Y,D),
    getFOLEffects(A,D,T,T2).
getFOLEffects(A,D,[X&Y|T],T2):-
    \+member2(X,D),
    \+member2(Y,D),
    getFOLEffects(A,[X,Y|D],[X,Y|T],T2). %
getFOLEffects(A,D,[X&Y|T],T2):-
    member2(X,D),
    \+member2(Y,D),
    getFOLEffects(A,[Y|D],[Y|T],T2).
getFOLEffects(A,D,[X&Y|T],T2):-
    \+member2(X,D),
    member2(Y,D),
    getFOLEffects(A,[X|D],[X|T],T2).

getFOLEffects(A,D,[X|T],[X|T2]):-
    \+(X=(P=>Q)),
    \+(X=(P&Q)),
    getFOLEffects(A,D,T,T2).
```

The parameters of `getFOLEffects` are (in order): the action performed to reach the current scenario, the provisional database including all the effects established by the `getModEffects` procedure, the list of elements in the provisional database still to be examined by this procedure, and the elements of the new database established by this procedure. The values of this new database are established by decomposing non-atomic properties of the provisional database. For example, if `X&Y` is an effect of performing the previous action, the elements `X` and `Y` are added to the database, assuming they are not already present.

`member2` Procedure

```
member2(Goal,DB):-
                member(Goal,DB).
member2(Goal1 & Goal2,DB):-
                member2(Goal1,DB),
                member2(Goal2,DB).
member2(Goal1 or _,DB):-
                member2(Goal1,DB).
member2(_ or Goal2,DB):-
                member2(Goal2,DB).
member2(Goal1=>Goal2):-
                member2(\neg(Goal1) or Goal2,DB).
member2(neg(Goal),DB):-
                \+member2(Goal,DB).
```

This procedure searches the database to see if a specific goal is present. This extends the predefined `member` procedure by searching within non-atomic predicates. If the goal is non-atomic, the procedure searches for the component parts.

`sortConflicts` Procedure

```
sortConflicts(A,DB,[],[]).
sortConflicts(A,DB,[moving(X,Y)|T],T2):-
        member(modal(A,moving(X,Z)),DB),
        \+(Z=Y),
        sortConflicts(A,DB,T,T2). %Remove if old  value

sortConflicts(A,DB,[moving(X,Y)|T],[moving(X,Y)|T2]):-
        \+member(modal(A,moving(X,Z)),DB),
        sortConflicts(A,DB,T,T2).
        %Store if no conflicting value.

sortConflicts(A,DB,[moving(X,Y)|T],[moving(X,Y)|T2]):-
        member(modal(A,moving(X,Y)),DB),
        sortConflicts(A,DB,T,T2). %Store if new value.

sortConflicts(A,DB,[H|T],[H|T2]):-
        \+(H=moving(X,Y)),
        sortConflicts(A,DB,T,T2).
```

As we discussed in Section 6.3.3, the `sortConflicts` procedure is required for managing the consistency of the scenario database. If a property holds as an effect of the most recent action and this conflicts with a property already in the database, the older property is removed. However, if the older conflicting property is part of the specification, then the newer property is removed.

`removeNegs` Procedure

```
removeNegs(_,_,[],[]).
removeNegs(SP,DB,[neg(P)|T],T2):-
      member(P,SP),
      write('Unable to contradict Specification Property'),
      nl,
      removeNegs(SP,DB,T,T2).
removeNegs(SP,DB,[P|T],T2):-
\+(P=neg(Q)),
member(neg(P),SP),
write('Unable to contradict Specification Property'),
nl,
```

```
        removeNegs(SP,DB,T,T2).

removeNegs(SP,DB,[P|T],T2):-
        \+(P=neg(Q)),
        member(neg(P),DB),
        \+member(neg(P),SP),
        removeNegs(SP,DB,T,T2).
        %If a previous scenario property has
        %been negated, it is removed.

removeNegs(SP,DB,[neg(P)|T],T2):-
        \+member(P,SP),
        removeNegs(SP,DB,T,T2).

removeNegs(SP,DB,[P|T],[P|T2]):-
        \+member(neg(P),DB),
        \+(P=neg(Q)),
        removeNegs(SP,DB,T,T2).
```

We discussed earlier the use of the neg operator to model explicit negation. This procedure models negation by removing all properties from the database that are also explicitly negated. All the explicit negations are also removed from the database.

addPermissions **Procedure**

```
addPermissions(DB, NewDB):-
    %If there is an immediate obligation
    member(io(X),DB),
    alterIPs(X,DB,NewDB).
addPermissions(DB,NewDB):-
    %if there is no immediate obligation
    \+member(io(X),DB),
    addIPs(DB,NewDB).
```

This procedure determines the P-structure of the current scenario. The alterIPs procedure is called if there is an immediate obligation in the current scenario. All actions are then prohibited except for the obliged action, which is immediately permitted. The addIPs procedure is called when there are no immediate obligations in the current procedure. All weakly permitted actions are immediately permitted.

addPrescriptions **Procedure**

```
addPrescriptions(DB,DB):- \+member(norm,DB).
```

```
addPrescriptions(DB,DB2) :-
        member(norm,DB),
        addMods(DB,DB,DB2).
```

This procedure defines an action description for each action, describing the effect on the normative constant of performing that action in the current scenario. This procedure encapsulates the definition of EDAL rule E13, given above.

addMods **Procedure**

```
addMods([action(X)|T],DB,[modal(X,norm)|T2]):-
        member(ip(X),DB),
        addMods(T,DB,T2).
addMods([action(X)|T],DB,[modal(X,neg(norm))|T2]):-
        \+member(ip(X),DB),
        addMods(T,DB,T2).
addMods([H|T],DB,T2):-
        \+H=action(X),
         addMods(T,DB,T2).
addMods([],DB,DB).
```

The addMods procedure adds action descriptions describing the effect that performing an action has on normativity. The effect depends on the existence of an immediate permission for that action in the current scenario.

6.4.2 Example Trace

There has been no formal verification that the interpreter correctly implements EDAL. This represents a considerable amount of work and we did not feel the verification was necessary for our purposes. We have been able to informally check that the interpreter returns the expected results for the examples tested. As the interpreter uses a separate database from Prolog, it is relatively simple task to validate the contents of the database in each scenario. The example trace works through two actions only, based on the procedures for halting trains given in the Channel Tunnel fire report. We have incorporated some extra rules to demonstrate different aspects of the interpreter. As such, the example does not demonstrate the procedures as given in the report.

Recall that the `behaviour` procedure takes a trace of actions, `actions`, a system specification, `initSpec`, and the current scenario database, which is evaluated from the concatenation of `initSpec` and `initScenario`. For this example, these properties are defined as follows:

```
initSpec([
```

```
            action(init),
            action(stop(t1)),action(decouple(t1)),
            (moving(t1,null))&(neg(norm)) =>o(evacuate(p)),
            modal(stop(t1), moving(t1,null)),
            modal(stop(t1), io(decouple(t1)))]).
initScenario([
                norm,
                p(init),
                modal(init,neg(p(init))),
                moving(t1,null)=>p(decouple(t1)),
                moving(t1,west),
                modal(init,io(stop(t1))),
                neg(p(decouple(t1))),
                moving(t2,null)]).
actions([init,stop(t1),decouple(t1)]).
```

Following: init

The action, `init`, is used to reach the scenario from which we will examine the behaviour of the system. Note that the list of properties that hold in each interval does not include negative values. These are implied by their absence:

```
Following:init
    Spec: [
          action(init), action(stop(t1)),
          action(decouple(t1)),
          moving(t1,null)&neg(norm)=>o(evacuate(p)),
          modal(stop(t1),moving(t1,null)),
          modal(stop(t1),io(decouple(t1)))]
    NewDB: [
          modal(init,neg(norm)), modal(stop(t1),norm),
          modal(decouple(t1),neg(norm)), action(init),
          action(stop(t1)), action(decouple(t1)),
          moving(t1,null)&neg(norm)=>o(evacuate(p)),
          modal(stop(t1),moving(t1,null)),
          modal(stop(t1),io(decouple(t1))), norm,
          modal(init,neg(p(init))),
          moving(t1,null)=>p(decouple(t1)),
          moving(t1,west), modal(init,io(stop(t1))),
          io(stop(t1)), moving(t2,null),
          ip(stop(t1)), modal(stop(t1),neg(io(stop(t1))))]
```

Following: stop(t1)

Following the performance of action `stop(t1)`, a number of these properties have changed. The first properties in the new database describe the normative effects of performing actions. The action description `modal(stop,neg(norm))` replaces the `modal(stop,norm)` from the initial scenario, because `stop(t1)` is no longer permitted or obliged. The action description `modal(decouple(t1),neg(norm))` has become `modal(decouple(t1), norm)` due to the presence of the action description `modal(stop(t1),io(decouple(t1)))` in conjunction with the predicate `moving(t1,null)=>p(decouple(t1))`. These rules also led to the deontic statements `io(decouple(t1))`,`p(decouple(t1)`,and `ip(decouple(t1)` holding. The specification properties remain constant. The scenario is still normative because `stop(t1)` was immediately obliged. `io(stop(t1))` no longer holds due to the rule `modal(stop(t1),neg(io(stop(t1))))`:

```
Following: stop(t1)

Spec: [
        action(init), action(stop(t1)),
        action(decouple(t1)),
        moving(t1,null)&neg(norm)=>o(evacuate(p)),
        modal(stop(t1),moving(t1,null)),
        modal(stop(t1),io(decouple(t1)))]
NewDB: [
        modal(init,neg(norm)), modal(stop(t1),neg(norm)),
        modal(decouple(t1),norm), action(init),
        action(stop(t1)), action(decouple(t1)),
        moving(t1,null)&neg(norm)=>o(evacuate(p)),
        modal(stop(t1),moving(t1,null)),
        moving(t1,null), modal(stop(t1),io(decouple(t1))),
        io(decouple(t1)), norm,
        modal(init,neg(p(init))),
        moving(t1,null)=>p(decouple(t1)),
        p(decouple(t1)), modal(init,io(stop(t1))),
        moving(t2,null), modal(stop(t1),neg(io(stop(t1)))),
        ip(decouple(t1)),
        modal(decouple(t1),neg(io(decouple(t1))))]
```

Following: decouple(t1)

Again, the specification properties remain constant. Although `decouple(t1)` was obliged in the previous scenario, it can still be performed and lead to a normative scenario because it is weakly permitted. `io(decouple(t1))` no longer holds, due to the rule `modal(decouple(t1),neg(io(decouple(t1))))`:

```
Following: decouple(t1)

Spec: [
       action(init), action(stop(t1)),
       action(decouple(t1)),
       moving(t1,null)&neg(norm)=>o(evacuate(p)),
       modal(stop(t1),moving(t1,null)),
       modal(stop(t1),io(decouple(t1)))]
NewDB: [
       modal(init,neg(norm)), modal(stop(t1),neg(norm)),
       modal(decouple(t1),norm), action(init),
       action(stop(t1)), action(decouple(t1)),
       moving(t1,null)&neg(norm)=>o(evacuate(p)),
       modal(stop(t1),moving(t1,null)),
       moving(t1,null), modal(stop(t1),io(decouple(t1))),
       norm, modal(init,neg(p(init))),
       moving(t1,null)=>p(decouple(t1)), ip(decouple(t1)),
       p(decouple(t1)), modal(init,io(stop(t1))),
       moving(t2,null), modal(stop(t1),neg(io(stop(t1)))),
       modal(decouple(t1),neg(io(decouple(t1))))]
```

6.5 Discussion of the DALEX Interpreter

Implementing the DALEX interpreter addressed well documented problems, such as the representation of negation in Prolog and the effects of programming with side effects (see Section 6.5.3). In addition, novel issues have been raised, such as how to model the accessibility relation of an action-based language using Prolog. The use of Clocksin and Mellish's formal translation techniques on the EDAL operator semantics helped clarify what these operators should and do say, both in EDAL and in Prolog. This section discusses the current interpreter and highlights areas where it should be improved.

6.5.1 Extending the DALEX Interpreter

The interpreter is extremely limited, in comparison to EDAL. There are numerous extensions required to enable it to model a larger subset of EDAL.

Agents can be quite easily added to action descriptions so that DALEX would more closely model EDAL. Other improvements, such as modelling reinvocation of immediate obligation and conflict in action descriptions would require more work. The current implementation of weak obligation does not represent the temporal aspect of its definition, and this requires either quantifiers or temporal operators. Temporal extensions to DALEX are possible, and are discussed below. However, the lack of quantifiers in the current interpreter needs to be addressed.

With finite sorts, the use of universal quantification does not create problems for the list based definition of the scenario database. However, attempts to implement quantifiers have been hampered by Prolog unifying the variables in the quantification statements in the scenario database.

Although the EDAL rules were formally converted to clausal form, the declarative implementation of the DALEX interpreter has required these clausal rules to be reinterpreted into Prolog procedures. This informal reinterpretation reduces the effectiveness of using DALEX to validate EDAL operators. Furthermore, it cannot be used as a proof tool. The effectiveness of this interpreter is, therefore, still quite low.

The interface of the interpreter can be significantly improved; the interpreter currently has only a simple 'pretty printing' procedure to improve the presentation of the databases. The incorporation of a parser, such as presented by Booth [Boo87], would make the interpreter a tool for model construction as well as analysis. Typographical errors, in particular, would be identified much faster. However, the clarity of the model and of the language remains the foremost factor in improving validity [GH96].

The predetermined trace of actions in the current implementation is adequate for prototyping the semantics of the operators and for small case studies. However, it is relatively simple to extend this approach to allow the user to dynamically select the action to be performed next. The ability to step through the scenarios and dynamically choose the behaviour trace should improve the user's understanding of the effect of the behaviour in each scenario [DFAB93]. The added flexibility would reduce the distinctions between formal accident modelling and the existing high level approaches to accident simulation.

6.5.2 Conflicts

One aim of modelling accident reports is the identification of inconsistencies. The requirements of automation can obscure this information. In the interpreter, if one property contradicts another, it is assumed to overwrite that contradictory value in the scenario database. Distinguishing inconsistencies from valid information change requires well documented output. For example, when the direction of movement of a train changes, the system should inform the user of this change and perhaps also of the rules that led to this change. Although the interpreter does give some feedback, extending the interpreter with a full explanation shell would provide a more complete description of the interpreter's behaviour.

6.5.3 Database Management: Declarative and Procedural Approaches

Prolog can be viewed as a declarative language: statements given in a Prolog program describe entities and existing relationships between these entities. Answering queries

is not based on telling the program what to do, but on interrogating it about what is true or false.

Prolog also contains extralogical predicates that give the programmer added control over the program execution by producing procedural side effects. However, these effects cannot be retracted during backtracking and thus can lead to inconsistency. The predicate `write (X)` is one example: it always succeeds, but has the non-retractable side effect of outputting `X` to the specified output stream. For example:

```
direction(X),
write('Train 1 is moving in direction'),
moving(t1, X),
write(X).
```

In this clause, the 'Train 1...' phrase could be written as many times as there are directions, as the interpreter will backtrack and try a different value of `X` if `moving(t1, X)` fails. Other extralogical predicates include `assert` and `retract`, which add and remove clauses and properties from the Prolog database.

The scenario database is currently managed declaratively, using a separate database and inference engine, although we exploit the `write` predicate to examine the contents of the scenario database. An alternative implementation manages the database procedurally, using Prolog's internal database and inference engine, and the extralogical predicates, `assert` and `retract`. The initial implementations of the interpreter took the latter approach.

The advantage of this approach is that Prolog supplies much of the inferential machinery of the interpreter. The disadvantage is that the interpreter is more difficult to check, formally or informally. The use of these extralogical procedural predicates is often viewed as poor practice. They are 'not logically consistent' [Mos86], and their use is often 'the result of intellectual laziness and/or incompetence' [SS86].

By comparison, it is a relatively simple task to validate the contents of the scenario database in the declarative interpreter, as the database is an explicit parameter in the interpreter.

6.5.4 Open and Closed World Approaches

The closed world assumption is relatively easy to model in Prolog; Negation as Failure makes the open world assumption much more difficult to model, as it requires the ability to explicitly negate properties. We used the interpreter to compare how these assumptions affect our ability to reason over scenarios. The interpreter clarified that the closed world assumption was more suitable for modelling and reasoning about natural language. These experiments with DALEX led to EDAL being developed with a closed world assumption.

Axioms
(Ax1) status(D,approved)
(Ax2) status(D,approved)→ P(ctsa, approve_safety(D))
(Ax3) [ctsa, approve_safety(D)]ctsaApproval(D)

Proof

status(D,approved) *(Ax1)*

.
.....:.....

¬ status(D,approved) **(closed)** | P(ctsa, approve_safety(D)) *(Developed Ax2)*
[ctsa, approve_safety(D)]ctsaApproval(D) *(Ax3)*

[ctsa, approve_safety(D)]:======= *New Tableau*
ctsaApproval(D)

Figure 6.2: Extract from Tableau-Based Proof

6.5.5 Alternative Interpreters

Developing or adapting an interpreter for a deontic action-based language is a major task. We considered a number of interpreters and theorem provers, before choosing to implement the interpreter in Prolog. The primary selection criteria were the degree of adaptation required to interpret EDAL and the training time required. Initially, we sought to adapt either the Prolog-based MAL animator [Boo87] or a tableau-based theorem prover for MAL [AC91] (see figure 6.2), produced by the FOREST project. Unfortunately, the definitions of the deontic operators in both tools differ significantly from those of EDAL. In the tools, the deontic operators are used to constrain, rather than prescribe, behaviour. This precludes the ability to model and reason about non-normative behaviour. As discussed, we were also interested in using the interpreter to explore the choice between using the open and closed world assumption in EDAL. DAL uses the former, the above tools use the latter. We anticipated that extending a more generic interpreter would make it easier to explore both options.

The generic theorem provers Isabelle [Pau94], HOL [GM93], and PVS [SOR93] were rejected due to the significant training time required. Temporal logic interpreters were considered, as the temporal logic 'next' operator reflects the temporal aspect of the action description operator. However, where the action description operator describes a property that holds following the performance of a particular action by a particular agent, the 'next' operator only describes a property that holds in the following state.

Tokio [AFMO86] is a temporal logic programming language, based on Moszkowski's Local Interval Temporal Logic(ITL) [Mos86] and incorporating ITL's 'next' operator for referencing the following interval. The interpreter for Tokio [KAFT86] is written in Prolog. We initially developed DALEX in Tokio, using 'next' to model movement between scenarios. This work helped clarify the semantics of EDAL to the extent that the Tokio interpreter was no longer necessary and the DALEX interpreter could

be defined solely in Prolog, as discussed above. Using Prolog without Tokio greatly facilitates the operator validation, as there is significantly less logic machinery in use during interpretation.

The difference between the Prolog and Tokio based interpreters is very slight: the interval logic recursive call to the `behaviour` procedure uses the Tokio 'next' operator (`@behaviour`). If EDAL is extended with more complex temporal operators, it may be useful to be able to add the extra temporal framework of Tokio to the interpreter.

6.6 Summary

This chapter has presented an interpreter for a deontic action-based language. The main task for this interpreter was to model the accessibility relation and the operator semantics of EDAL. This work was initiated by the desire to automate and examine the semantics of the EDAL operators. Many other issues arose from the attempt to translate EDAL semantics into the Prolog-based interpreter and these were discussed in detail. Finally, the weaknesses of the DALEX interpreter were discussed and suggestions were made for improvements to the interpreter.

Part IV

Concluding Remarks

Chapter 7

Future Directions

This section describes further extensions to and applications for the work presented in this thesis.

7.1 Alternative Applications

The modelling process can be applied to other types of report and in other areas of the system design lifecycle.

7.1.1 Legal and Insurance Reports

Accident reports are also produced for litigation and insurance purposes. Deontic logic has a history of legal application, and its ability to represent both prescriptions and descriptions of behaviour may make it better suited to modelling such reports. On the other hand, the utility of the formal model would be limited as these inquiries are less driven by technical tests and more by committee judgements under legal rules [BR91].

7.1.2 Modelling Numerous Sources

In this thesis, we have concentrated on building a model of a single document. Further information could be drawn in from other sources, to assist the construction of a more accurate model. Other sources could be witness statements, newspaper reports, alternative reports[1], or a domain expert. An additional advantage of this is the potential identification of hidden or unintentional inconsistencies, as demonstrated in [JT96].

[1]For example, as well as the public inquiry into the Channel Tunnel fire by the CTSA, the French Judiciary conducted a criminal investigation and Eurotunnel conducted an internal investigation.

As with inconsistencies within the report, it is important to locate and highlight such inconsistencies. In Section 7.4.3, we outline methods that could assist with this task.

7.1.3 Accident Report Construction

Accident inquiries already produce numerous complex models to examine various aspects of the accident. For example, the Channel Tunnel fire investigation used virtual reality models to assist the reconstruction of the incident [SS97b].

We assert that incorporating the construction of a formal deontic action-based model into the existing accident reporting process would lessen the identified problems with the natural language reports, aid the structuring and consistency of the argumentation, and improve the accuracy of the information in the report.

This activity would use the same techniques as we have presented in this thesis. However, the emphasis would be less on the analysis and criticism of the reports. Instead, the natural language and argumentation problems highlighted by the formalisation could be remedied prior to publication. Accident reports use techniques, such as timelines, photographs, and graphs, to emphasise or illustrate particular information. Some of the more accessible output from the SCS steps, such as the natural language descriptions and the agent hierarchy, could be added to the report to highlight particular aspects of the information in the report. This could also improve the structuring of the report, emphasised by users as a weakness of existing reports [SJ99]. The quality of the formal model would also improve; with access to the full information from the inquiry, fewer assumptions would be required.

Another possible use for the model at this stage is in modelling drafts of the report recommendations, which can then be examined in the context of the accident scenario. This would facilitate the study of how the implementation of these recommendations could affect the system behaviour. For example, the Channel Tunnel fire report model could be slightly altered to examine how the behaviour of the system would differ if the Eurotunnel management ensured their staff were adequately trained and experienced. Such predictive models would require significantly more detail about the system than is presented in the report.

Automating the production of a correct formal model from an informal document is, at present, infeasible. However, the reverse task of producing natural language documents from formal specifications is possible (for example, [Kem87]). The Prolog parser for MAL, given in Booth [Boo87], provides a good initial point for developing such a tool for EDAL. A natural language translation of the formal model would improve the accessibility of the model and could also be used as a basis for a consistent, unambiguous accident report.

7.1.4 System Design and Model Reuse

The role of accident analysis in the system design cycle is to inform current and future designs. The current process of producing natural language reports, based on an investigation, to inform future designs is rather indirect. The accident analysis method described in this thesis uses systems design techniques and there is the potential for at least some of this information to be directly passed into the system (re)design stage.

The design of safety critical systems is a highly structured activity and it is difficult to anticipate the capacity for integrating the deontic action model or its results. One possibility that would exploit the mutual focus on normativity is to use the norms defined in the model as a basis for a requirements document. The requirements document produced for a system could also be compared with the normative constraints defined in the model. Lutz [Lut97] describes how formal models can be reused to validate the requirements of similar systems.

Although the information given in the report is essentially deterministic, we have shown that information about general behaviour can be extrapolated. With a model of the general behaviour, it is possible to analyse other potential behaviour in addition to the accident behaviour. For example, a general model of the behaviour in the Channel Tunnel fire report could be used to examine whether the accident would still have occurred if the procedures of the system were not deficient. This work bears strong similarities to the development of safety cases. A safety case is 'a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment' [BB98]. The inclusion of such safety cases in the design process helps avoid the large costs associated with the construction of unsuitable designs. A model of general behaviour enables the analyst to examine fault tolerance and system reactions to well understood non-normative behaviour. Studying the interface between normativity and non-normativity facilitates the identification of non-normative behaviours and other potential hazards, similar to more traditional hazard analysis techniques such as Hazards and Operability Analysis (HAZOP).

At an abstract level, there are similarities between the behaviour and requirements of distinct but related systems. The thermodynamic model of the fire in the King's Cross fire report, which uncovered the 'trench effect' of flame movement, has general ramifications for all building designs incorporating escalators and other trench structures with inclined surfaces. An EDAL model of the general system behaviour could be used to analyse the behaviour of other similar systems. For example, the King's Cross fire report model could be applied to other underground train systems, such as the Paris metro. However, as we stated above, such predictive models require more information about the system than an accident report provides.

7.2 Continued Development of EDAL

The intention of this work was to use the existing DAL language. Designing, changing, or extending a language involves extra work and potentially leads to an overly complex language. However, as we discussed earlier, the definition of the operators of DAL did not seem well suited to accident report modelling.

7.2.1 Alternative Operators

EDAL has a relatively small set of operators. However, it is still under development and further case studies may highlight the need to change or add operators.

Alternative definitions for the existing operators may be appropriate, if it is felt that they cannot adequately represent the norms in the report. The definition of the immediate obligation operator in EDAL was influenced by the North Anna reactor incident (see Appendix B.3). We initially followed the view of the later DAL publications that obligation and permission should be separate, as it seemed that the North Anna operators were obliged, but not permitted, to prevent the endangerment of the plant. The philosophical discussion of whether 'ought' implies 'can' is ongoing [Mar95]; see Jones' [Jon87] response to McCarty [McC86]. However, on closer examination, the North Anna incident actually enforces the more traditional view that obligation does imply permission. Organisationally, the operators of North Anna had neither an obligation nor permission to shut down the reactor from the NRC, yet evidently they held some other view that obliged, and thus permitted, this. EDAL's ability to model the norms of the agents is very useful for understanding the reasons for such conflicts.

Additional operators can be added to EDAL to improve the ease with which certain concepts are expressed. For example, consideration was given to a 'strong permission' operator, which denotes an action that, once permitted, is always permitted. Such an operator provides additional information to the reader. In this case, that the permission will never be revoked, However, EDAL is also made more complex by the introduction of additional operators. After considering this trade-off and the low propensity of 'strong permission' statements required in the accident report models examined, the operator was omitted. Other prospective operators were rejected for similar reasons.

In some cases, the operators under consideration were felt to be too complex. For example, Khosla [Kho88] proposed an obligation operator that allows sets of actions to be obliged. We decided to use the more traditional and basic definitions first, to see if these were adequate.

7.2.2 Alternative Modalities

Although the focus of this thesis has been on the role of deontic modalities and modalities of action within accident reports, other aspects of accident reports have been identified as being of interest, such as time and belief (for example, Johnson and Telford

[JT96], Johnson [Joh97b], Ladkin and Loer [LL98]). It is likely that some accident reports would benefit from formal models that are able to express yet other modalities, such as: desires, wishes, necessity, contingency, possibility. Additional operators can be added to EDAL to extend the language with further modalities.

Epistemic logic facilitates reasoning about how the human agents in the accident system behave in response to internal beliefs about the situation they are in. Modelling these beliefs provides an important insight into their behaviour, particularly where the behaviour would otherwise have been modelled non-deterministically. For example, in EDAL if an agent made a mode error and performed a non-normative action because of this, it would be viewed as having acted non-normatively. Epistemic modalities could enable a more accurate study of where, and why, the mode error occurred. The work of van Linder et al. [vLMvdH97] has extended a language, similar to EDAL, with epistemic modalities, and modalities for expressing wishes and desires. Fisher et al. [FWD96] have demonstrated that, although difficult, developing proof methods for such multimodal logics is possible.

Temporal logic facilitates detailed reasoning about the temporal sequence of events. A simple implementation of time is possible, in an EDAL model, using an integer count function as a system constraint. For example, given some integer value for time t and an integer x, $t = x \rightarrow [A, \alpha]t \geq x$.

The action logic of van Linder et al. includes a second action description operator from dynamic logic, of the form $\langle\alpha\rangle\phi$. This states that performing an action α *may* result in ϕ holding. This may have uses for representing ambiguity and incompleteness in the model. In addition, it would be useful for representing actions with uncertain outcomes.

7.2.3 Development Considerations

Further work is needed to examine the trade-off between ease of expression and added complexity for the readers of the model, before further operators or modalities are added.

There are techniques (i.e., Why-Because Analysis [LL98]) that enable many further aspects of accident reports and systems to be modelled, such as epistemic and real-time information. Models constructed using EDAL necessarily have a tighter focus on fewer system aspects, and are thus less complex and easier to review.

The major contribution of formal methods and languages in any application is in increasing confidence. Increases in the level of detail or in the number of aspects of the system to be modelled increase model complexity, and ultimately have diminishing returns in confidence, particularly if the modelling language has poor scalability [MP91]. Furthermore, a specification that is intended to model many aspects of a system can be inconvenient if the user wishes to focus on a specific aspect. The analyst must be aware of this when choosing a modelling language and a level of abstraction.

7.2.4 Extending the Current Tool Support

The DALEX interpreter must be extended so that it formally models the semantics of all the EDAL operators, supports dynamically defined action traces, and has a more accessible interface. As discussed in section 3.5.2, further tools are required to increase the usability of EDAL and SCS.

In particular, an information management tool would be extremely helpful. During the production of the case study, ensuring that terms were used consistently was a very time consuming task. This was because it required searching within and between numerous documents. Existing requirements engineering tools, such as DOORS 4, offer significant help in this respect. The numbers of agents and actions identified in the report are within the bounds of most software engineering projects, so we believe that these tools would cope with managing the information extracted from the report.

A critiquing system for accident report models would be another automated means to evaluate and improve the output of the modelling process. Critiquing systems have been created for many diverse domains from kitchen design [FLMM91] to theorem proving [AG97]. Critiquing systems use a series of predefined heuristics and requirements to assess a particular artefact. Feedback on the criticisms produced by the system can be fed back into the system to enrich the knowledge base. However, as we discuss in Section 7.7.2 below, we first need to build a knowledge base of the requirements, metrics, and heuristics for accident report models. A similar system could also be developed for critiquing accident reports themselves.

7.3 Alternative Languages

We described earlier, in Sections 5.4.2 and 5.4.3, some possible alternatives to EDAL. These include other deontic logics and other accident report modelling notations.

One of EDAL's main weaknesses for accident modelling is its poor scalability. Botting and Johnson [BJ97] demonstrate that a modular language, in this case Object Z, can improve usability and scalability of accident report models. However, these models share another of EDAL's weaknesses, as they are rather complex and inaccessible to non-logicians.

The use of carefully structured text has been suggested as an alternative approach to overcoming the problems identified in accident reports. The advantages of this approach lie in its ease of use for non-logicians. Although there have been no comparative analyses in the accident report modelling literature, structured text has been shown to be less capable than formal methods for modelling complex systems [LHHR94]. In addition, the use of a textual notation does not provide the numerous complementary views of aspects of the system that SCS does.

In Section 5.4.2, we suggest localised normativity as a means to avoid the limitations of EDAL's notion of normativity. Conceivably, this localisation could form part of an

object-oriented logic, similar to Object-Z [DKRS91] and Structured MAL [FGKQ92]. The combined benefits of such an approach would be improved scalability from the object-oriented approach, improved accessibility and information management through the use of a structured method, and a more intuitive notion of component normativity. The production of such a logic would be a significant task for a skilled logician and is beyond the scope of this thesis.

Defeasible logics also offer a more expressive and intuitive notion of normativity. Furthermore, there is a large amount of current work in the field of defeasible logic (for example, Nute [Nut97]). As we stated earlier, the information these logics require makes it difficult to construct a defeasible model of current accident reports. However, this information can be derived from prospective accident prevention techniques, such as incident reporting schemes. This information has been used to examine the role of cognitive factors in task performance (e.g. Busse and Johnson [BJ99]). However, this work only examines erroneous behaviour. Defeasible logic could complement this work by modelling the numerous, potentially conflicting norms that can influence operator behaviour. Although this would be a very interesting avenue to explore, it is also beyond the scope of this thesis.

7.4 Alternative Methods

Earlier, we described how some of the steps in the SCS method could be altered or supported with tools. However, SCS is a number of years old. The use of a new requirements engineering method may be a simpler, more effective way to improve the formalisation process than adding tool support to the SCS steps. Indeed, many of these new methods have existing tool support.

7.4.1 Very Structured Common Sense

Industrial trials of SCS and MAL highlighted the problem of scalability and later FOREST work focussed on modularity. Very Structured Common Sense was developed during the latter stages of the FOREST projects, to add object-oriented structuring to the method [LJ91]. Very Structured Common Sense promotes modular development (for example, using E-R diagram hierarchies) and specification. Unfortunately, tool and method support for Very Structured Common Sense is limited and, as it was developed for use with Structured MAL, it is not integrated with (E)DAL.

7.4.2 Accident Analysis Method

Before adopting SCS, we initially developed a more application-oriented methodological framework that incorporates hierarchical task analysis and Hollnagel's classification of errors. As with SCS, the intermediary steps of this method use common, easy

to understand techniques that guide the development of the model. The first step of the model uses Hierarchical Task Analysis (HTA) to build a prescriptive model of the system behaviour. The second step of the method uses Hollnagel's classification of erroneous actions [Hol93] to categorise the errors documented in the report and to identify other potential errors. The output of these steps feed into the third step, the EDAL model.

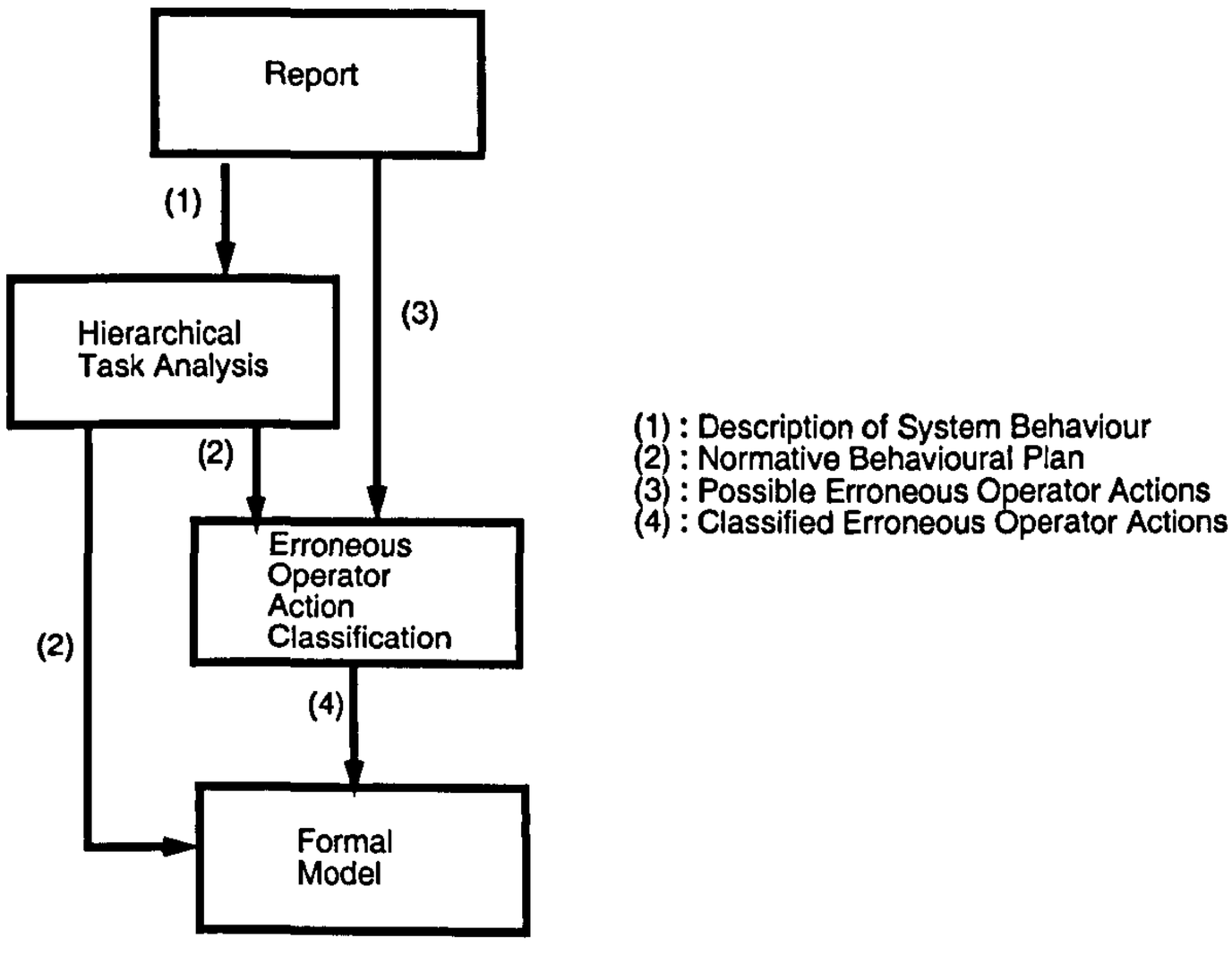

Figure 7.1: Framework for the Formalisation of Accident Reports

In comparison to SCS, the method suffers from poor traceability between the steps. The elicitation and modelling process is much less gradual than with SCS, making the modelling task more difficult and the output more likely to contain errors.

This framework is more suited to system (re)design, using the accident report to inform the error classification. The error classification technique is not fully exploited by accident report modelling, as too little information is given in the report to determine how the system would react to other potential non-normative behaviours identified by the error classification.

The use of task analysis and error classification techniques is mirrored in other error identification techniques such as Task Analysis-Linked EvaluatioN Technique (TALENT) [Rya88] and Sneak Analysis [Hd91].

7.4.3 Viewpoint-oriented Methods

A *viewpoint* is defined as the combination of an agent and the view held by the agent [FS96]. The P-structure for a particular agent, the set of obligations, permissions and prohibitions that hold in that scenario, can be modelled as part of a viewpoint.

Viewpoint modelling is a large area of research in requirements engineering. A number of requirements engineering methods, such as Viewpoints [FKNG92] and PREview [SS97a], explicitly model different viewpoints and address the problems of inconsistency and conflict between viewpoints. The use of a viewpoint-oriented requirements engineering method would enable contradicting norms, such as those present in the North Anna incident, to be represented and resolved.

There are two ways viewpoint-oriented methods could be used in accident report analysis: the differing views of the writers can be modelled or viewpoints representing the agents involved in the accident can be modelled.

Report Analysis

Chapters in accident reports are often written by experts in certain fields, such as forensics, thermodynamics, and meteorology. The differing views of the accident held by these writers are represented, implicitly or explicitly, within the body of the report. The inconsistencies are not restricted to differences of opinion about the prescriptive behaviour of the system, and may even incorporate differing views on the actual events and their importance [JT96]. A writer's representation of an accident can be affected by the level of involvement in the accident, knowledge of the system, and the writer's background [Lep87a, Lek97].

Using a viewpoint-oriented method, the different views can be examined individually or compared, inconsistencies and ontological overlaps identified, and suggestions made for reconciling conflicts [SF97]. Even without reconciliation, the inconsistency would be explicit and could be justified by the writers involved.

This approach could also be used for identifying inconsistencies between different source documents. In some cases, the writers of the source documents openly disagree about the accident. For example, the pilot of Airbus A320-100 F-GFKC disagreed with the report into its crash at Habsheim on the 26th of June [MoP89] and published a book outlining his view of the accident [Ass92][2]. In such instances, highlighting the inconsistencies could help focus on the disagreements and the justification for the differing views.

[2]The differences between these reports are examined in Mellor [Mel94], from which this example is drawn.

Accident Analysis

The view of an agent can include a notion of the normative behaviour of other agents. Again, views of different agent's P-structures may not always be consistent. For example, the King's Cross fire report states that the London Underground Management did not view itself as obliged to implement the policy of contacting the Fire Brigade immediately when any fire occurred. It is evident that the report writer and the fire brigade did. This is shown on page 76 of the report in a letter from Deputy Assistant Chief Officer Kennedy of the Fire Brigade to Mr Cope, the Operations Director (Railways), which states:

> I am gravely concerned to find that, contrary to the professional advice of the (Fire) Brigade, a two-stage procedure has been introduced for notifying the Brigade of fire occurring on the London Underground railway system. Following the recent fire at Oxford Circus underground station, the Brigade made it quite clear that the Brigade should be called immediately to any fire on the underground network.

Conflicts in an agent's behaviour prescriptions, the prescription source, and an agent's view of external prescriptions provide important information for understanding the behaviour exhibited in the system. Viewpoint-oriented methods could be used to represent and analyse the views of the distinct agents within the system. However, accident reports rarely contain enough information about the agent's views for this to be feasible. In addition, reports reflect their writers' interpretation of the events. The construction of accurate viewpoints for the different agents would probably be more successful if it were conducted during an inquiry. Additional psychological expertise is required if prescriptive models of the agents in the accident are to be developed, not least due to the complexities of the human consciousness regarding beliefs, obedience, attitudes, social interaction, and recall, particularly in stressful situations [Gle91]. As with the report writers, the witnesses' interpretation of the accident is also affected by the level of involvement in the accident and knowledge of the system [Lep87a].

7.4.4 Other Methods for Modelling Accidents

There already exist many varied methods for modelling accidents and accident reports (see Benner [BR85] for examples). It is not an aim of this thesis to evaluate or criticise these approaches. As we have stated earlier, the work presented here is intended for use in conjunction with these other methods. We have demonstrated that one method developed for use in a software engineering domain can be applied to the domain of accident analysis. An avenue for future work is a further exploration of other software engineering based analysis techniques, such as contextual enquiry [BH97] and OOSE [JCJÖ92], and whether these could be advantageously applied in the domain of accident analysis.

7.5 Implementation Issues

Failure to develop the interface between a new formal method and existing technology is a common flaw in the integration process [Sta93]. Having demonstrated the value of EDAL and SCS, we need to address the feasibility of integrating this technology into the existing reporting framework of accident reporting. Weber-Wulff [WW93] gives a number of practical properties that a formal method should have for it to be successfully introduced into industry. Some of these properties, such as teachability, have been addressed in this document. However, there are further considerations that are lacking in all the accident report modelling literature. These include ensuring the method has adequate documentation, ensuring that it is possible to confine, and if necessary, reverse the introduction of the method, and ensuring that the intrusion caused by the method and tools is minimised.

Other issues that are identified as hindering the use of formal methods, such as poor tool support and the separation of information from formality [CGR93], have been addressed, to a certain extent, during the development of EDAL.

7.6 Quality Management Issues

Although we have illustrated the benefits EDAL and SCS might bring to accident report modelling, improved reports are not guaranteed by the introduction of formal methods. Kneuper [Kne97] describes how formal methods are effectively useless to (software) developers who do not have a *quality system*, a measure of the standard of quality management and assessment approximate to that described in ISO 9000 [HW94]. However, currently few industries possess adequate quality management in their accident reporting process to fully benefit from the introduction of the formal models. For example, the report construction guidelines of ICAO (International Civil Aviation Organisation) prescribe format and content information, but include no guidelines for ensuring the quality of the report [ICA97]. In the case of the Channel Tunnel fire report, there were no guidelines on its construction, although the co-rapporteurs each had significant experience of preparing or reviewing reports. In conjunction with expert advisors, they structured and scoped the information in the report. The report then evolved into the document published through a process of review. As members of the Channel Tunnel Safety Authority, the co-rapporteurs cannot be viewed as independent from the regulatory body. The independence of the reporters is an important factor in engendering trust in the outcome of the investigation [Rea97].

Problems exist throughout the process of identifying the accident causes, extrapolating recommendations from these causes, communicating these recommendations to the relevant parties, and modifying the appropriate systems to benefit from these recommendations [Lev95]. In many cases, the introduction of formal methods to the reporting process is only one of numerous improvements that can be made to improve the quality of accident reports.

7.7 Evaluation

7.7.1 Evaluation of SCS and EDAL

As we stated in the introductory chapter, this work is not intended to immediately fit into the existing accident reporting process. We have presented an initial study of an accident report modelling method in use. Using the Software Engineering Institute's Capability Maturity Model [Hum89], the method presented would be at the lowest level of maturity. Increases in the level of maturity lead to more effective and predictable processes that produce products that meet the needs of the user. However, the maturity of the presented method will only increase if future applications of the method place much greater emphasis on project management and change control.

Increases in the maturity of the method facilitate more consistent resource and cost estimates. Such estimates are produced using established evaluation techniques, such as COCOMO II [BCH+95]. The COCOMO II post-architecture model requires significant empirical data of the processes involved to produce accurate cost, effort, and schedule estimates. The COCOMO II early design model is much more applicable to the current status of the method, but, at this time, this model has yet to be calibrated.

The Cognitive Dimensions framework [GP96] is another evaluation technique that could currently be applied to EDAL and the SCS method. It does not use any specific metrics and the criteria and output are designed to be comprehensible to non-specialists. Cognitive Dimensions could be used to provide an initial evaluation of EDAL and SCS in use, although it is intended for use in conjunction with other evaluation techniques.

Benner [BR85] also presents a set of criteria by which accident models can be evaluated. This evaluation is informal and subjective, but the findings demonstrated that a "large degree of ... improvement" was possible by choosing a highly rated model.

The following chapter presents an informal evaluation of EDAL and SCS, based on the work presented here.

7.7.2 Evaluation of Formal Accident Analysis

This thesis has given some convincing arguments for the inclusion of agents and actions in behaviour-oriented models of an accident, as well as the inclusion of deontic statements for reasoning about organisational failure. However, other accident models, such as a first order logic model of software code [Tho94], do not feature any of these concepts. We have also highlighted the relative benefits and costs associated with a 'heavyweight' formal modelling approach, a 'lightweight' formal modelling approach, and a semiformal modelling approach to accident report modelling.

It is our view that before assessing further languages, extending existing languages or integrating alternative methods for accident report modelling, there should be a retrospective appraisal of the work in this field to date. A significant number of languages,

methods, and accidents have been examined, and it should now be possible to empirically determine:

- if particular languages or presentations are suited to modelling certain types of accident.
- if there are aspects of the system that should feature in all accident report models.
- if the costs of using certain languages and methods outweigh their advantages.

The evaluation criteria can be gathered from the various publications presenting these methods and languages. For example, in this document we have highlighted the need for traceability, unambiguous semantics, and tool support. Related studies, such as Kirwan's appraisal of human error identification techniques [Kir98], Benner's rating of accident models and investigation methodologies [BR85], and Lonchamp's assessment of process modelling methods [Lon94], can be used to assist the generation of relevant evaluation criteria. Gathering this information together will provide a vital reference for designers and analysts looking to model particular aspects of the behaviour of an accident.

Chapter 8

Conclusions

This thesis has addressed practical concerns in the domain of accident report modelling. We have demonstrated that accident report models can support designers in their use of accident reports. In particular, we have demonstrated that the combined use of EDAL and SCS facilitates the identification and examination of key objects, entities, behaviour, and argumentation within the report.

Depending on the modelling needs, the method could be applied in a number of different ways In this chapter, we briefly describe some envisaged application categories, followed by a more detailed envisaged application scenario. We then present an informal analysis of the contributions of this work. Without more tightly controlled experiments, a more accurate evaluation of the language and method is not possible. However, the use of a real accident report in the case study increases our confidence that the results are relevant to an industrial application.

8.1 Scenarios of Use

As we discussed in Section 1.2.3, we anticipate the presented method being used as a post-hoc accident report modelling tool by either analysts or teams of analysts and designers.

8.1.1 General Usage Categories

The interests of the modeller(s), their level and area of training, and the amount of resources available are all factors determining the information that will be modelled and the modelling technique that will be adopted. Given these factors, we can describe four general accident report modelling categories of usage for the SCS method.

Semi-Formal Modelling

In the first category, the steps of SCS are applied to modelling the report. However, the final formal modelling step is omitted. This approach may be taken when there are no analysts available to the modelling team, or when there is a low level of resources available. This approach can also be taken as an initial step, with the option of performing the formal modelling steps later if it is deemed necessary.

There are numerous benefits this approach offers. Minimal training will be required, as each step uses well-known engineering techniques. The output from each step provides a novel and accessible view of a particular subset of the information in the report. This facilitates the communication of this information to non-technical users of the model. The information in each view can also be traced back to its source in the report, making it easy to validate. The method provides early payback, in that the first iteration of each step provides significant information about the report. If a wide-scope approach to the modelling process is taken, the modeller(s) will gain a strong understanding of the report. This facilitates the generation of new hypotheses about the content of the report. The elicitation process is also very successful at identifying ambiguities, errors, and incompleteness in the report.

The disadvantage of this approach is that, relative to more formal techniques, little confidence can be placed in the findings. Furthermore, there is no way to examine sequences of actions and thus to see the effects that this behaviour has upon the state. The use of semi-formal analysis steps also makes it possible that there may still be some ambiguity.

'Lightweight' Formal Modelling

The second category reduces the possibility of ambiguity by also including the formal modelling step. This approach might be chosen if the motivation for the modelling process was concerned with ensuring the relevant content of the report was represented as thoroughly and as accurately as possible. The approach requires a trained logician and therefore cannot be as flexibly applied as the previous approach.

We describe this approach as a 'lightweight' formal approach, as it does not involve the high cost activity of manual proof. Such approaches are gaining increasing popularity in software engineering (for example, Jackson and Wing [JW96] and Easterbrook et al. [ELC+98]). This 'lightweight' approach to formal modelling brings together all the information elicited in the SCS model steps in one concise representation of the accident report. This is useful for examining the model, although it requires training in the formal notation. Furthermore, this concise representation makes informal reasoning about the model relatively straightforward in comparison to reasoning about the information in the report.

As with the semi-formal model, the information in the formal model is traceable back to the earlier steps of SCS and to the report itself. Given a wide scope approach in

the initial SCS steps, further formal models can be efficiently produced from the semi-formal model.

'Heavyweight' Formal Modelling

In the third usage category, manual proof techniques are used to gain additional confidence in the theorems of the model. This approach may be taken when informally reasoned theorems must be as accurate as possible, and if there is no tool support for the reasoning process. This process again requires a trained logician and can be time intensive.

As demonstrated, an EDAL formal model can be used to demonstrate errors of omission, qualitative failure, and insufficient prescriptions, that exist as a result of sequences of actions. Unfortunately, as well as requiring considerable resources, large manual proofs carry a high probability of error.

Modelling with Tool Support

In the fourth category, tools exist for each modelling step. The cost of producing tool support can be very high: industrial requirements engineering packages are expensive and producing new object languages for 'heavyweight' theorem provers, such as Isabelle, requires expertise and time. At relevant points in this thesis, we have outlined ideal and existing tools that could contribute to the modelling process.

This fourth approach requires that the tool user be trained in the use of the tool as well as the underlying technique(s). Development and integration costs may be high, but with a fixed method, the maintenance costs will be reasonable. Furthermore, the addition of tool support to any of the previous categories of use will bring significant benefits. In the first approach, traceability and visualisation would be improved. In the second and third approaches, animation could be added to enable sequences of behaviours to be examined more accessibly. In the final approach, the introduction of an interpreter, model checker, or theorem prover, would improve confidence in theorems.

Level of Formality	Users	Cost	Benefits	Possible Tool Support
Semi-Formal	All	Low-Medium	Immediate Payback, Good Visualisation, Low Training, Information Management, Traceability	RE Tools, SE Tools for Individual Steps
Lightweight	Requires Logician	Medium	High Confidence, Sequences Modelled, Single Concise Representation	Executable Languages (State Charts)
Heavyweight	Requires Logician	High	Very High Confidence	Interpreter, Model Checker, Theorem Prover

8.1.2 Detailed Usage Scenario

The above descriptions give generic examples of how the method could be used. In this section, we give a detailed scenario of the method in industrial use. This scenario is hypothetical, but is intended to give the reader a more concrete example of how the method would fit into the accident report analysis process.

Scenario Background

Following a public accident inquiry, a report is published that contains a number of recommendations for company X that will prevent a recurrence of the accident and will improve the general safety culture of the company. The company has suffered adverse publicity following the accident and has already implemented some new safety measures in its system. In the interests of the company, the board of directors is keen to take immediate further action, based on the recommendations of the report.

This is one example of a realistic motivation for using the method and tools presented in the thesis.

Stage 1: Setting up the Analysis Project

The management wishes to explore fully the recommendations of the report. These recommendations will guide the design of future safety mechanisms of the system. The design will be an expensive exercise with significant public interest, and thus it is vital that the recommendations are not misinterpreted. Past experience with using accident reports has made the management wary of the contents, and they feel that the report should be checked for errors and inconsistent information. The management therefore chooses to fund an analysis project that will use a structured method to model the

report content. As this is a new approach to examining reports, this project will run in parallel with more established techniques.

Ideally, this project would run prior to the more established techniques so that outcome of the modelling process can inform the redesign process before any design decisions have been made.

The decision to model an accident report is based on the level of importance attached to the accuracy, consistency, and completeness of the information within the report, balanced against the level of 'trust' the users place in the contents. Current users of accident reports were found to have at least 'natural critical scepticism' and at most 'outright mistrust' for the conclusions [SJ99]. These users use the accident report for background detail and then conduct separate experiments to gain the information they require. If the information in the report is adequate then this is a very inefficient and costly approach. The method presented here is a less costly approach to assessing the contents of the report. In the given scenario, the management intend to comply with the report but are taking certain precautionary steps to ensure that the information given in the report is accurate, complete, and consistent.

Focus of Analysis

The upper management wishes the entire report and all the recommendations to be semi-formally modelled. Through discussions with the redesigners of the system and safety consultants, they have decided that their main concerns lie in improving the company's safety culture. This aspect of the system has already been addressed to a certain extent by the initial reaction of the company to the accident. In addition, the redesigners envisage that meeting the recommendations as laid out in the report would entail undoing some of the earlier safety improvements. Safety consultants employed by the company feel that the recommendations of the report require excess change to the system. They feel that these changes will be very costly and that there will be no tangible safety benefit resulting from these recommendations.

The safety culture recommendations in the report require a restructuring of the middle management of the company. One of the aspects the analysis of the report will address is examining how the previous managerial safety framework contributed to the accident. If possible, the analysis will also examine whether the proposed restructuring would prevent such an occurrence happening.

The focus of the analysis is a strong determinant on the method that is used. In this case, the accident report analysis method outlined in this thesis is well suited to modelling the sequences of behaviour and the concept of managerial failure within these sequences. The choice of method will also impact on the choice of personnel for the team. In this case, at least one member of the team chosen should be trained in the application of the SCS method and the EDAL language.

Focussing the analysis in this way inevitably will introduce some bias in the model. To a certain extent this is unavoidable, as we wish to model an abstraction of the report. However, as we discussed earlier, a wide-scope approach to modelling can assist in reducing bias. Furthermore, with this approach the modelling team do not need to know the focus of the analysis in the semi-formal modelling stage. This will also

further reduce the level of analyst bias in the model.

Level of Analysis
The board of management decide that at least one logician should be included in the modelling process. This decision was taken because the board intends the analysis to examine how the highlighted managerial failures impact upon the behaviour in the accident.

The first level of analysis is the semi-formal modelling step. The team requires logicians for the light and heavyweight formal modelling options that follow. After the method has been employed in a number of cases, the probability of the semi-formal modelling process leading to formal modelling will be more accurately predictable. If the analysis is intended solely to improve the structure the information contained in the report and gain new overviews of certain aspects of the information, a logician is not necessary.

Team Composition
The allocated team size for this project is limited to one, an EDAL trained logician. The decision to use a logician was taken based on the expectation that the project would require some formal modelling. The decision to use an EDAL trained logician was based on the focus of the analysis being on middle-management issues. The decision to limit the team to a single member was based on the management's view that the problems highlighted in the report would not require a very complex model and on the fact that the method is being used experimentally by the company.

The resource and personnel allocation for the analysis project has important repercussions for the project, although some aspects of these decisions are flexible and can be reassessed during the course of the project. The composition and size of the team devoted to the analysis process will affect the speed, efficiency, and accuracy of the results produced.

As we have outlined elsewhere, these teams would be made up of domain experts and logicians. If these formal steps are a realistic possibility, then a logician should be in the team from the start, as the semi-formal modelling process is a good way to learn about the content of the report. Furthermore, having a logician perform the semi-formal modelling step ensures that at the end of this step, lightweight and heavyweight modelling steps can be implemented at minimum cost. Domain experts ideally should not be directly involved in modelling the report content, as their knowledge can lead to assumptions being made about implied information. In choosing to use solely a logician, the board have enabled the extent of the modelling process to be flexible.

We have no empirical evidence of a team-based application of SCS in this domain, although it proved effective in its intended domain [LJ91]. However, we will outline the issues that might lead to larger teams being required. We envisage that further members would be added for a complex modelling task. Furthermore, if the focus of the analysis was broader, there may be the need to hire other logicians to model more than one modality. Logicians in place at the start of the project need not be skilled in the use of theorem provers or model checkers. If these tools are available, an appropriately skilled logician can be recruited to the team if and when this level of rigour is deemed

necessary. Theorem proving and model checking do not require familiarity with the report. Domain experts do provide useful information in the process but they only play a supporting role. One approach would be to have a domain expert who is not part of the modelling team but meets with them on a regular basis to discuss issues that have arisen. Although we will concentrate on describing a one-person analysis scenario, we will also discuss, where appropriate, further issues that we envisage may affect a team-based analysis process.

Stage 2: Semi-formal Analysis

Once the project begins, the analyst works alone. During each step there are many instances of implied or ambiguous information located. In each of these cases, the analyst is required to assess the various alternative interpretations available. These are then documented and a decision is taken on which alternative will be modelled. The reasoning behind this decision is also documented.

We do not define a specific method for this decision making process. The decision will be made based on the analyst's knowledge of the report and the system. The decision can be altered later in the process to examine if and how the different possibilities affect the model.

In a team-based project, the domain experts would have significant input on ranking these alternatives and determining the most likely. However, the documentation of the alternatives should still be carried out, as subsequent analysis of the model is dependent on these assumptions.

Tool support during this stage assists the tracing the information elicited back to the model or to the appropriate decision. Other tools can be used to document the reasoning behind any decisions that need to be made.

The outcome of this stage is a series of accessible views of the information in the model. The logician prepares a report that provides the output of the semi-formal modelling process, along with the assumptions that underpin the models. The logician also includes an evaluation of the areas of uncertainty and ambiguity in the report that relate to the focus of the analysis. From performing the semi-formal modelling steps, the logician has accumulated an in-depth knowledge of the report content. One section of the analyst's report is devoted to hypotheses that the logician has generated from this knowledge. These should be ranked in order of relevance to the focus of the analysis.

The modelling process is being conducted to examine the sequences of behaviour that led to the accident. After the analyst has presented the report to a subset of the board of management, the redesigners, and the safety consultants, the decision is taken to formally model the report with a view to analysing the prescriptive and descriptive behaviour models.

Depending on the aptitude of the team and the complexity of the model, another team member could be added at this stage to assist the translation of the logical model to the

executable notation. However, in this scenario the logician remains the sole person on the project.

Regardless of whether the analysis continues or not, the findings of the semi-formal analysis are presented to the system redesigners. This information is useful for correcting and clarifying the information that they had elicited from the report. Furthermore, the redesigners may have accessed further sources of information and can check the accuracy of the report model and the assumptions that underlie it. The redesigners may require some training in the techniques used by SCS so that they can understand the output from the steps of the process. However, this should be minimal as most of these techniques are well-known engineering notations.

The analyst might also consider examining other sources of information and independently checking the information elicited from the report. If there are inconsistencies discovered, the analyst could request that tests be carried out on the system to check the accuracy of the conflicting information. If the information in the report is not deemed an accurate representation of the system, then the analyst could create a second model. This second model would be an updated version of the report model that enables the system to be analysed.

Stage 3: Lightweight Formal Modelling

Once the decision has been made to proceed with the lightweight formal modelling, a number of hypotheses should be generated. The scope of the model will depend largely on the hypothesis under investigation.

Using the general focus of the analysis as a starting point, the analyst generates a ranked list of hypotheses from discussions with the safety consultants and system redesigners. The analyst then constructs a formal model based on a particular hypothesis from the list. Once this is completed, the analyst is able to informally reason about sequences of behaviour in the model.

Similar discussions would take place in the team-based approach, both between the logicians and domain experts and between the logicians and the redesigners of the system. In both cases, the decision process should again be fully documented.

If the results that are generated from this informal reasoning have significant repercussions, formal reasoning techniques should be used to confirm the validity of the reasoning. This might be the situation if the conclusions or recommendations of the report are deemed invalid.

As we have described in this thesis, tool support can assist with visualising the informal reasoning process.

Stage 4: Heavyweight Formal Modelling

The informal reasoning demonstrated that a relatively trivial procedural change would

avoid the same hazard as the costly hardware based fault tolerant systems recommended by the report. The potential savings offered convince the management to begin preparing an appeal. However, before the company approaches the legislating body, tests must be carried out to ensure that the informal reasoning is valid. The assumptions underlying the formal model are sent to further domain experts for scrutiny. As there are no theorem provers and model checkers currently available for EDAL, the analyst produces a manual proof of the theorem. Another expert user of EDAL is then contracted to review the proof.

The situation where a model checker or theorem prover is available would obviously have been much preferable, due to the potential for mistakes in manual proof.

Once the formal proof has been reviewed, a series of system tests are performed to provide physical evidence for the claims. A report is then prepared for submitting to the legislating body. This document argues the case for the procedural change. It includes the opinions of safety experts, the physical test results, and details of the proof and its underlying assumptions.

The formal proof provides a high level of confidence in the theorem. This helps to ensure that the costly tests that are required to provide physical evidence are only performed when they will provide the expected results. Currently, adding the proof to the submitted report is unlikely to be much more than a rhetorical aid. However, if it is examined it will heighten the confidence of the reader in the argument of the report. This situation could change greatly if the situation occurred where a formal proof played a key role in influencing an appeal or a legal action. Such an explicit demonstration of the value of formal reasoning would be very likely to encourage their use and acceptance.

Conclusion

Although we have described a scenario where the heavyweight formal modelling step is performed, in current practice this situation is unlikely to arise. System designers do not use accident reports this rigorously at present. This situation is rather peculiar as the contents of accident reports are vital step in the accident analysis process and have significant weight in the legal framework. If an accident recurs and the organisation involved has failed to act on past recommendations, extremely damaging litigation could follow. If an accident recurs and the organisation had acted upon the recommendations, the reputations of the organisation involved and the organisation that produced the report will be badly damaged and legal action could follow. The processes of developing and using of accident reports need to change if situations like this are to be avoided. This situation motivated the work presented in this thesis.

Although the steps of the method are well-defined, many aspects of its application in this domain require further development. In the given scenario, we outline areas where decisions are required. Until further investigation is performed, we cannot elaborate on the methods and heuristics that will help determine the appropriate action. We also outline areas where information can transfer between the modelling process and the redesign process. However, the integration of the method with the existing redesign

process is still immature. Further case studies will facilitate the refinement of the integration of these processes.

Therefore, the adoption of the method is likely to depend on refinements to the method, changes in the way accident reports are developed and used, and an increase in industrial acceptance of the benefits of formal modelling and reasoning. Until then, as we have described, semi-formal and formal modelling can still offer major benefits to the examination and demonstration of properties in the report.

8.2 Examining the Thesis Contributions

We have described briefly how differing modelling requirements can result in the work described in this thesis being applied in different ways. The main case study of this thesis explored a usage scenario in which a trained logician builds and manually reasons about a formal model. We now present an assessment of the contributions of this work.

8.2.1 Methodical Information Elicitation and Model Construction

This thesis describes the first application of a methodological approach to accident report modelling. The need for such an approach has been less obvious in previous accident report models as they generally model only small fragments of the report argumentation. However, the information required for these fragments may be scattered throughout the report. The large size of most accident reports makes it difficult to manage the information they contain. A structured modelling approach would therefore be beneficial, regardless of the size of the model being produced or the language in which it is built.

The major benefit of using SCS is that it guides the elicitation of information and the development of the model in incremental, manageable steps. In addition, the stepwise approach also adds a degree of traceability between the model and the report. The readability of the model is improved, both by this traceability and by the use of well-known engineering techniques. These techniques require little training, making them useful for communicating information to those outside the modelling process and between members of the modelling group.

As discussed earlier, many errors of many types were found during the construction of the model. This is a strong argument for the value of the SCS method. Each step of the method focuses on particular aspects of the report in isolation. This facilitates the identification of omissions and errors in the report, and thus leads to a more accurate model. Although this thesis has only demonstrated the use of SCS, the methodical analysis of a document, using any technique, would be likely to uncover some errors.

Continued use of SCS helps identify further, more complex errors in the report. However, the immediate payback the method brings makes it a useful strategy, even when

formal modelling is not the aim. The strength of the requirement for accuracy in the model will determine how SCS and EDAL are used.

The construction of the formal model from the semi-formal model is not necessary, although, as we outlined above, there are situations where it can bring additional benefits. As EDAL formal models are relatively straightforward to construct from the SCS semi-formal model, the poor scalability of EDAL is less of a problem. Many small EDAL models can be constructed instead of a single large one. Furthermore, from the semi-formal model it should be possible to produce other complementary formal models in different formal languages.

8.2.2 A Novel Wide Scope Modelling Approach

Experience with developing the Channel Tunnel fire report model suggests that the accident report modelling process requires careful thought and attention, but can ultimately reveal complex information about the system that is not apparent from reading the report.

In this thesis, we have demonstrated that by taking a holistic approach to the initial semi-formal modelling steps a more complete and accurate model can be produced. This approach requires a methodological modelling process, due to the large amount of information elicited. Furthermore, this approach can take significantly longer to produce a formal model. However, one of the major benefits of producing a wide scope semi-formal model of an accident report is that numerous formal models can be efficiently created from it. This approach therefore has clear advantages for a modelling team that wishes to examine more than one fragment of the report behaviour.

This approach is also very useful for educating the analyst on the content of the accident report. As we discussed in Section 1.2.3, the analysts who construct the model have little domain knowledge, thus reducing the chance of analysis bias. However, as they construct the model their knowledge of the report increases. This knowledge is useful for generating report specific hypotheses. In the usage scenario where a domain expert is producing a semi-formal model, this approach should also reduce modelling bias by requiring the modeller to examine the full text of the report.

We believe that the wide scope approach should always be employed where the content of the report is under serious investigation. This approach is particularly useful for identifying relevant information that is implicit or located far from the main body of the relevant information. Furthermore, this approach is useful in locating inconsistencies. Analysts using the conclusion analysis modelling approach will stop searching when they find information that matches their search criteria. However, the report may also contain other contradictory information. With the wide scope approach, both sets of information would be uncovered.

Despite the additional costs of the wide scope approach, we feel that it offers significant advantages to any non-trivial examination of the content of an accident report.

8.2.3 Reuse of Requirements Engineering Technology

We have already highlighted the major advantages that a methodological modelling approach can bring to accident report modelling. One of the most interesting aspects of the application of SCS was that it was successfully applied, with little adaptation, in a different domain to the one it was designed for.

The success of SCS has a number of benefits for the field of accident report modelling. There has been little published work on eliciting information from accident reports. Leite [Lei87] claims that ignoring elicitation issues in the field of requirements engineering impeded the field for a whole decade. In exploiting the similarities between requirements engineering and accident report modelling, we benefit from many years of experience in the requirements engineering field.

SCS was developed for use in industry and underwent a number of industrial usability trials. The steps of SCS are based on well known and usable software engineering techniques. Although they are being used in a new domain, we argue that all engineers have similar skill sets and that learning these techniques should not present any significant problems. In using SCS, we therefore benefit not only from the field of requirements engineering, but also from the field of software engineering.

8.2.4 Supporting Formal Accident Analysis Claims

The field of accident report analysis is concerned with improving the accessibility and accuracy of accident reports. Previous applications of formal languages to modelling accident reports have identified numerous problems with structure, argumentation, and presentation of accident reports. In Section 1.1.7, we identified a large number of these problems. We have demonstrated that the methodical construction of a formal model of the report identifies and makes explicit many of the argumentation weaknesses and improves the visualisation of the information in the report.

In this thesis, we have also addressed a number of more practical concerns with the use of formal methods. Many industries still have misgivings about using formal methods due to the costs involved. There is a common perception that formal methods are used only in 'heavyweight' approaches. Due to the current level of mistrust with the contents of accident reports, it is difficult to imagine a scenario where 'heavyweight' formal methods would be used in industry to model accident reports (particularly without tool support).

In line with much recent formal methods work, we advocate a lighter weight approach to accident report modelling. As discussed in Section 5.4.3, there are a number of ways that the behaviour in the report can be examined that do not require the rigour of a formal proof. Using a less formal approach, such as building an executable state chart model, has significant benefits over manual proof, such as improved visualisation. Although automated proof tools reduce the cost of manual proof, they require expert users. Less formal approaches are typically easier to learn and to use.

Other practical concerns addressed include EDAL's poor scalability and accessibility.

Scalability

The problem of the poor scalability of EDAL has been highlighted throughout this thesis. In conjunction with an engineering-style method and careful structuring, it is possible to construct models of large systems. However, the poor scalability still affects the readability and maintenance of a large model. As stated above, this problem is lessened if numerous small formal models can be efficiently produced. If an object-oriented logic is used, the problems of scalability could be reduced further.

Accessibility

A common criticism of text-based formal languages is their poor accessibility. To a non-logician, an EDAL specification is unlikely to be immediately understandable. Generous commenting of the formal model, in conjunction with the more accessible views produced in the SCS method, should produce an accident report model that can be understood by logicians and non-logicians alike.

8.2.5 Modelling Organisation Constraints

EDAL is not 'competing' against other proposed techniques for accident report analysis. Indeed, the differing focuses of the other techniques would complement each other, when used in conjunction. The choice of formal modelling language should be largely dependent on the information that is to be examined.

If the modellers are interested in the duties and obligations that shaped the behaviour of the operators, or in examining the point at which these obligations were broken, then a deontic logic is a suitable formalism. EDAL's deontic operators offer a novel prescriptive view of the report. As well as enabling normative reasoning, they separate action descriptions from prescriptions. These are generally ambiguously mixed together in the report, which can confuse the reader's interpretation of the system and the behaviour described. EDAL's treatment of actions and agents as 'first class citizens' is well suited to the domain of accident modelling, in which pairings of actions and agents are the 'building blocks' [BR92].

Accidents occur due to the combination of both active failures and latent conditions. There is growing interest in the latent conditions caused by organisational and managerial failure in accidents. As we have demonstrated, EDAL's latent failure operators extend the notion of normativity in EDAL models. This enables the models to be used to examine and reason about the organisational and managerial prescriptions in the system, such as the managerial safety framework.

EDAL is the first formal language that has been designed to model latent failure. In the case study, we demonstrated that reasoning about latent failure and non-normative actions can highlight interesting properties of the report and system. However, although EDAL enables latent failures and conditions to be modelled, reasoning about normativity in accidents is still constrained by the binary notion of normativity of deontic logic.

EDAL models do not have to be used in a 'heavyweight' manner. For example, informal reasoning can be used to examine the theorems given in Chapter 5. The ability to reason about normative and non-normative behaviour is advantageous for accident analysis. However, we feel that the ability to represent movement between normative and non-normative scenarios would be exploited far better in reasoning about design problems, such as common failure types and fault tolerance. In an accident report, little information is given about more general behaviour, thus restricting the reasoning that can be done about the system.

8.2.6 Tool Support for Validating EDAL Operators

The need for better tool support for EDAL is common to most formal methods [Sta93]. However, lack of tool support has been identified as one of the main factors that discourage the uptake of formal methods in industry [AP93].

The DALEX interpreter proved to be a useful tool for validating the semantics of the language and the behaviour of the model. However, the lack of a formal translation process between DALEX and EDAL means that we cannot have high confidence in the output of the interpreter.

Even as an informal tool for animating the EDAL semantics, DALEX is limited. DALEX only executes a subset of the EDAL operators and does not support universal quantification. This restricts the complexity of behaviour that the interpreter can be execute. The example of DALEX in use is fairly trivial. With larger models and action traces, the state information becomes more difficult to track, thus reducing its value as a visualisation tool.

The interpreter has a limited use, in modelling the semantics of the EDAL operators and providing a means to informally examine sequences of behaviour in the model. However, it remains unclear whether the DALEX interpreter could ever formally execute EDAL and we feel that a new tool should be developed if a more rigorous tool is required. Adapting an existing modal object language of one of the general purpose theorem provers would provide a more effective proof tool.

8.3 Summary

Deontic logics have been applied to many diverse fields. This thesis has demonstrated that representing and differentiating between prescriptions and descriptions of

behaviour can help the formal analysis of an accident report. This work has also given strong arguments for the representation of actions and agents in accident report models. The thesis has backed up previous work criticising the current reporting process and demonstrated that constructing and reasoning about formal accident report models highlights problems in reports. The numerous benefits of employing an engineering-style method for eliciting information from the report and guiding the construction of the model have been identified. Finally, the issue of tool support for the method and language was addressed. The benefits of any formal approach to accident report modelling will be limited unless there is adequate methodological and tool support.

The work presented in this thesis has answered many questions and has uncovered many further interesting issues. Although formal and structured methods may not be used to model accident reports in the near future, the work in this area convincingly highlights the improvements that are possible in an aspect of safety engineering that has, until recently, been overlooked.

Bibliography

[AC91] W. Atkinson and J. Cunningham. Proving properties of a safety-critical system. *IEE Software Engineering Journal (Special Issue on Safety-Critical Systems)*, 6(2):41–50, March 1991. Charles Babbage Award Paper.

[AFMO86] T. Aoyagi, M. Fujita, and T. Moto-Oka. Temporal logic programming language Tokio: Programming in Tokio. In E. Wada, editor, *Logic Programming '85: Proceedings of the 4th Conference*, volume 221 of *Lecture Notes in Computer Science*, pages 128–137, Tokyo, Japan, July 1986. Springer-Verlag.

[AG97] J.S. Aitken and P. Gray. Providing relevant feedback by critiquing. In Y. Bertot, editor, *Proceedings of the Third International Workshop on User Interfaces for Theorem Provers (IUITP'97)*, pages 1–8, INRIA-Sophia-Antipolis, Cote D'Azur, France, September 1997.

[All97] R. Allison. *Inquiry into the fire on heavy goods vehicle shuttle 7539 on 18 November 1996.* Her Majesty's Stationary Office, London, 1997.

[AP93] S.M. Austin and G.I. Parkin. Formal methods: A survey. Technical report, Division of Information Technology and Computing. National Physical Laboratory, Teddington, Middlesex, United Kingdom, March 1993.

[Ass92] M. Asseline. *Le Pilote est-il coupable?* Les Éditions Numéro Un, Paris, 1992.

[BB98] P.G. Bishop and R.E. Bloomfield. A methodology for safety case development. In *Industrial Perspectives of Safety-critical Systems (Proceedings of the Sixth Safety-critical Systems Symposium)*, Birmingham, UK, February 1998. Springer-Verlag.

[BBD+96] J. P. Bowen, R. W. Butler, D. L. Dill, R. L. Glass, D. Gries, A. Hall, M. G. Hinchey, C. M. Holloway, D. Jackson, C. B. Jones, M. J. Lutz, D. L. Parnas, J. Rushby, J. Wing, , P. Zave, and H. Saiedian. An invitation to formal methods. *IEEE Computer*, 29(4):16–30, April 1996.

[BCH+95] B. Boehm, B. Clark, E. Horowitz, R. Madachy, R. Shelby, and C. Westland. Cost models for future software life cycle processes: Cocomo 2.0. *Annals of Software Engineering*, 1:57–94, 1995.

[Ben97] L. Benner, Jr. *Introduction to Investigation*. Stillwell, Oklahoma State University Fire Protection Publications, 1997.

[BH97] H. Beyer and K. Holtzblatt. *Contextual Design: Defining Customer-Centered Systems*. The Morgan Kaufmann Series in Interactive Technologies. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1997.

[BJ97] R.M. Botting and C.W. Johnson. A formal and structured approach to the use of task analysis in accident modelling. Technical Report TR-1997-31, Department of Computing Science, University of Glasgow, October 1997.

[BJ99] D.K. Busse and C.W. Johnson. Human error in an intensive care unit - a cognitive analysis of critical incidents. In *Proceedings of the Seventeenth International Conference of the Systems Safety Society, Orlando, FL, USA, August 1999*, 1999.

[BJT97] C.P. Burns, C.W. Johnson, and M. Thomas. Agents and actions: Structuring human factors accounts of major accidents. Technical Report TR-1997-32, Department of Computing Science, University of Glasgow, October 1997.

[Boo87] J. P. Booth. Animating formal specifications. AERE Harwell Report R 12869, Computer Science and Systems Division, Harwell Laboratory, 1987.

[BR85] L. Benner, Jr and I.J. Rimson. Rating accident models and investigation methodologies. *Journal of Safety Research*, 16:105–126, 1985.

[BR91] L. Benner, Jr and I.J. Rimson. Quality management for accident investigations, part 1. *ISASI forum*, 24(3), October, 1991.

[BR92] L. Benner, Jr and I.J. Rimson. Quality management for accident investigations, part 2. *ISASI forum*, 25(2), March 1992.

[Bro87] F.P Brooks, Jr. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, pages 10–19, April 1987.

[Car47] T. Carlyle. *Critical and Miscellaneous Essays: Collected and Republished*, volume 1. Chapman and Hall, London, 3 edition, 1847.

[Car89] K.L. Carper, editor. *Forensic Engineering*. Elsevier, 1989.

[CBF+95] A. Coombes, L. Barroca, J.S. Fitzgerald, J.A. McDermid, L. Spencer, and A. Saeed. Formal specification of an aerospace system: the attitude monitor. In M.G. Hinchey and J.P. Bowen, editors, *Applications of Formal Methods*, International Series in Computer Science, chapter 13, pages 307–332. Prentice-Hall, 1995.

[CGR93] D. Craigen, S. Gerhart, and T. Ralston. Formal methods reality check: Industrial usage. In J.C.P. Woodcock and P.G. Larsen, editors, *FME'93: Industrial-Strength Formal Methods, Proceedings of the First International Symposium of Formal Methods Europe*, volume 670 of *Lecture Notes in Computer Science*, pages 250–267, Odense, Denmark, April 1993. Formal Methods Europe, Springer-Verlag.

[Chi63] R.M. Chisolm. Contrary-to-duty imperatives and deontic logic. *Analysis*, 24:33–36, 1963.

[CK92] M.G. Christel and K.C. Kang. Issues in requirements elicitation. Technical Report CMU/SEI-92-TR-12, ESC-TR-92-012, Software Engineering Institute, Carnegie Mellon University, 1992.

[Cla78] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*. Plenum Publishing Co., New York, 1978.

[CM87] W.F. Clocksin and C.S. Mellish. *Programming In Prolog*. Springer-Verlag, 3rd edition, 1987.

[Cum96] D.D. Cummins. Evidence for the innateness of deontic reasoning. *Mind and Language*, 11:160–90, 1996.

[CW96] E.M. Clarke and J.M. Wing. Formal methods: State of the art and future directions. Technical Report CMU-CS-96-178, Carnegie Mellon University, 1996.

[DeM78] T. DeMarco. *Structured Analysis and System Specification*. Prentice-Hall, Eaglewood Cliffs NJ, 1978.

[DFAB93] A. Dix, J. Finlay, G. Abowd, and R. Beale. *Human-Computer Interaction*. Prentice-Hall, 1993.

[DKRS91] R. Duke, P. King, G.A. Rose, and G. Smith. The object-z specification language. Technical Report 91-1, Department of Computer Science, University of Queensland, May 1991.

[Dun87] K.D. Duncan. Fault diagnosis training for advanced continuous process installations. In J. Rasmussen, K. Duncan, and J. Leplat, editors, *New Technology and Human Error*, chapter 19, pages 209–221. John Wiley and Sons, 1987.

[ELC+98] S. Easterbrook, R. Lutz, R. Covington, J. Kelly, Y. Ampo, and D. Hamilton. Experiences Using Lightweight Formal Methods for Requirements Modeling. *IEEE Transactions on Software Engineering*, 24(1):1–11, January 1998.

[Fen88] D. Fennell. *Investigation into the King's Cross Underground Fire*. Her Majesty's Stationary Office, London, 1988.

[FGKQ92] J. Fiadeiro, S.J. Goldsack, S. Kent, and W. Quirk. Design Issues in Structured MAL. Technical Report FOREST Research Deliverable Report WP1.R2, Department of Computing, Imperial College of Science, Technology and Medicine, London, January 1992.

[FKNG92] A. Finkelstein, J. Kramer, B. Nuseibeh, and M. Goedlicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1):31–58, 1992.

[FLMM91] G. Fischer, A.C. Lemke, R. McCall, and A.I. Morch. Making argumentation serve design. *Human-Computer Interaction*, 6:393–419, 1991.

[Flo67] R.W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical aspects of computer science: Proceedings of the American Mathematics Society symposia*, volume 19, pages 19–31. American Mathematical Society, 1967.

[FM91] J. Fiadeiro and T. Maibaum. Temporal reasoning over deontic specifications. *Journal of Logic and Computation*, 1(3):357–395, 1991.

[FP86] A. Finkelstein and C. Potts. Structured Common Sense: The elicitation and formalization of system requirements. In *Software Engineering '86*, pages 236–250. Peter Peregrinus, London, UK, 1986.

[FP88] A. Finkelstein and C. Potts. Building formal specifications using "Structured Common Sense". In *Proceedings of the 4th International Workshop on Software Specification and Design*, pages 108–114. IEEE Comp. Soc. Press, 1988.

[FS96] A. Finkelstein and I. Sommerville. The viewpoints FAQ. *Software Engineering Journal*, 11(1):2–4, 1996.

[Fuc92] N.E. Fuchs. Specifications are (preferably) executable. *Software Engineering*, 5(7):323–334, 1992.

[FWD96] M. Fisher, M. Woolridge, and C. Dixon. A resolution-based proof method for temporal logics of knowledge and belief. In *Proceedings of the International Conference on Formal and Applied Practical Reasoning(FAPR)*, volume 1085 of *Lecture Notes in Computer Science*, Bonn, Germany, June 1996. Springer-Verlag.

[FWH95] R.E. Fields, P.C. Wright, and M.D. Harrison. A task centred approach to analysing human error tolerance requirements. In *RE'95 - Second IEEE International Symposium on Requirements Engineering*, York, 1995.

[GF93] O.C.Z. Gotel and A.C.W. Finkelstein. An analysis of the requirements traceability problem. Technical Report TR-93-41, Department of Computing, Imperial College of Science, Technology and Medicine, London, 1993.

[GH96] A. Gravell and P. Henderson. Executing formal specifications need not be harmful. *Software Engineering Journal*, March 1996.

[Gle91] H. Gleitman. *Psychology*. W.W. Norton & Co., 3rd edition, 1991.

[GM93] M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving System for Higher-Order Logic*. Cambridge University Press, 1993.

[GP96] T.R.G. Green and M. Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual High Integrity Systems*, 7:131–174, 1996.

[Gur95] C.A. Gurr. Supporting formal reasoning for safety critical systems. *High Integrity Systems*, 1(4):385–396, 1995.

[Hal90] A. Hall. Seven myths of formal methods. In *IEEE Software*, pages 11–19, September 1990.

[Har84] D. Harel. Dynamic logic. In D.M. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic Vol. II*, chapter 10, pages 497–604. Reidel, Dordrecht/Boston, 1984.

[Har87] D. Harel. Statecharts: A visual formalisation for complex systems. *Science of Computer Programming*, 8:231–274, June 1987.

[Har88] D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, May 1988.

[Har96] J. Harrison. Formalized mathematics. Technical Report 36, Turku Centre for Computer Science (TUCS), Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland, 1996. Available on the Web at: `http://www.cl.cam.ac.uk/users/jrh/papers/form-math3.html`.

[HB95a] M.G. Hinchey and J.P. Bowen, editors. *Applications of Formal Methods*. International Series in Computer Science. Prentice-Hall, 1995.

[HB95b] M.G. Hinchey and J.P. Bowen. Applications of formal methods FAQ. In M.G. Hinchey and J.P. Bowen, editors, *Applications of Formal Methods*, International Series in Computer Science, chapter 1, pages 1–15. Prentice-Hall, 1995.

[Hd91] H.A. Hahn and J.A. deVries. Identification of human errors of commission using sneak analysis. In *Proceedings of the Human Factors Society 35th Annual Meeting*, pages 1080–1084, San Francisco, 2-6 September 1991.

[HK95] H. Herrestad and C. Krogh. Deontic logic relativised to bearers and counterparts. In J. Bing and O. Torvund, editors, *25 Years Anniversary Anthology in Computers and Law*, pages 453–522. Complex-Tano, Oslo, 1995.

[HLN+90] D. Harel, H. Lachover, A. Namaad, A. Pnueli, M. Politi, R. Sherman, A. Schtull-Trauring, and M. Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *IEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[HM95] Z. Huang and M. Masuch. An outline of ALX3, a multi-agent action logic. In *Proceedings of the 1995 Dutch Conference on Artificial Intelligence,(NAIC'95)*, 1995.

[Hoa69] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of ACM*, 21, October 1969.

[Hoe96] C. Hoes. Published on The Investigation Research Roundtable web pages at: `http://patriot.net/~luben/4exper.html#needs`, October 1996. Comment on Investigation Research needs.

[Hol93] E. Hollnagel. The phenotype of erroneous actions. *International Journal Of Man-Machine Studies*, 39:1–32, 1993.

[Hum89] W.S. Humphrey. *Managing the Software Process*. SEI Series in Software Engineering. Addison-Wesley, Reading , MA , USA, May 1989.

[HW94] D. Huyink and C. Westover. *ISO 9000*. Irwin Professional Publishing, New York, 1994.

[IBN96] U. Isaksen, J.P. Bowen, and N. Nissanke. System and software safety in critical systems. Technical Report RUCS/97/TR/062/A, Department of Computer Science, The University of Reading, UK, December 1996.

[ICA93] *Investigation of Human Factors in Accidents and Incidents*. Montreal, Canada, 1993.

[ICA97] *Accident Investigation Manual*. Montreal, Canada, October 1997.

[Jac83] M.A. Jackson. *System Development*. Prentice-Hall International, 1983.

[JCJÖ92] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering: A Use-Case Driven Approach*. Addison-Wesley, Reading, MA, 1992.

[JMW95] C.W. Johnson, J.C. McCarthy, and P.C. Wright. Using a formal language to support natural language in accident reports. *Ergonomics*, 38(6):1265–1283, 1995.

[Joh76] W.G. Johnson. Investigative methods useful in safety. In V.J. Pezoldt, editor, *Rare Event/Accident Research Methodology: Proceedings of a Workshop Held at the National Bureau of Standards*, Gaithersburg, Maryland, 26–28 May 1976.

[Joh92] C.W. Johnson. *A Principled Approach to the Integration of Human Factors and Systems Engineering for Interactive Control System Design.* PhD thesis, University of York, 1992.

[Joh97a] C.W. Johnson. Beyond belief: Representing knowledge requirements for the operation of safety-critical interfaces. In S. Howard, J. Hammond, and G. Lindgaard, editors, *Interact'97*, pages 207–212, London, United Kingdom, 1997. Chapman and Hall.

[Joh97b] C.W Johnson. The epistemics of accidents. In *International Journal of Human-Computer Studies*, volume 47, 1997.

[Joh97c] C.W. Johnson. Proving properties of accidents. In C.M. Holloway and K.J. Hayhurst, editors, *4th NASA Langley Formal Methods Workshop*, NASA Conference Publication 3356, pages 21–34, NASA Langley Research Centre, Hampton, United States of America, 1997.

[Joh99] C.W. Johnson. Why human error analysis fails to support systems development. *Interacting with Computers*, 11(5):517–524, 1999.

[Jon87] A.J.I. Jones. On the relationship between permission and obligation. In *International Conference on Artificial Intelligence and Law*, 1987.

[JT96] C.W. Johnson and A.J. Telford. Extending the application of formal methods to analyse human error and system failure during accident investigations. *Software Engineering Journal*, 11(6):355–365, 1996.

[JW96] D. Jackson and J. Wing. Lightweight Formal Methods. *IEEE Computer*, 29(4):22–23, April 1996.

[KAFT86] S. Kono, T. Aoyagi, M. Fujita, and H. Tanaka. Implementation of temporal logic programming language Tokio. In E. Wada, editor, *Proceedings of the 4th Annual Conference – Logic Programming '85*, volume 221 of *Lecture Notes in Computer Science*, pages 138–147. Springer-Verlag, 1986.

[Kem87] G. Kempen, editor. *Natural Language Generation: New Results in Artificial Intelligence, Psychology and Linguistics*, number 135 in NATO ASI Series E: Applied Sciences. Martinus Nijhoff, 1987. Proceedings of the NATO Advanced Research Workshop on "Natural Language Generation".

[Kho88] S. Khosla. *System Specification: A Deontic Approach.* PhD thesis, Imperial College of Science and Technology, University of London, 1988.

[Kir98] B. Kirwan. Human error identification techniques for risk assessment of high risk systems –part 1: review and evaluation of techniques. *Applied Ergonomics*, 29(3):157–177, 1998.

[Kle88] T. Kletz. Wise after the event. *Control and Instrumentation*, 20(10):57–59, October 1988.

[KM87] H.J. Komorowski and J. Maluszyński. Logic programming and rapid prototyping. *Science of Computer Programming*, 9:179–205, 1987.

[Kne97] R. Kneuper. Limits of formal methods. *Formal Aspects of Computing*, 9:379–394, 1997.

[Lei87] J.C. Leite. A survey of requirements analysis. Advanced Software Engineering Project Technical Report RTP-071, University of California at Irvine, Department of Information and Computer Science, 1987.

[Lek97] A.K. Lekberg. Different approaches to incident investigation – how the analyst makes a difference. *Proceedings of the 15th International System Safety Conference*, pages 178–183, 1997.

[Lep87a] J. Leplat. Accidents and incidents production: Methods of analysis. In J. Rasmussen, K. Duncan, and J. Leplat, editors, *New Technology and Human Error*, chapter 13, pages 133–142. John Wiley and Sons Ltd., 1987.

[Lep87b] J. Leplat. Occupational accident research and systems approach. In J. Rasmussen, K. Duncan, and J. Leplat, editors, *New Technology and Human Error*, chapter 17, pages 181–191. John Wiley and Sons Ltd, 1987.

[Lev95] N.G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1995.

[LG97] Luqi and J.A. Goguen. Formal methods: Promises and problems. *IEEE Software*, 14(1), 1997.

[LHHR94] N.G. Leveson, M.P.E. Heimdahl, H. Hildreth, and J.D. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, 20(9):684–707, September 1994.

[LJ82] B. Lepatner and S.M. Johnson. *Structural and Foundation Failures: A Casebook for Architects, Engineers, and Lawyers*. McGraw-Hill Book Company, New York, 1982.

[LJ91] J.A. Lynch and J.F.E. Johnson. Industrial exemplars in FOREST research: Requirements specification of an aircraft hydraulics system isolation valve controller. Technical Report NFR/WP4.1/BAe/RP/009 BAe-WIT-RP-RES-SWE-3038, October 1991.

[LJ97a] L. Love and C.W. Johnson. Using diagrams to support the analysis of system 'failure' and operator 'error'. In H. Thimbleby, B. O'Conaill, and P. Thomas, editors, *People and Computers XII: Proceedings of HCI'97*, pages 245–262, London, United Kingdom, 1997. Springer-Verlag.

[LJ97b] L. Love and C.W. Johnson. Using 'extended' fault trees in conjunction with traditional accident reports. In S. Howard, J. Hammond, and G. Lindgaard, editors, *Interact'97*, London, United Kingdom, 1997. Chapman and Hall.

[LL98] P.B. Ladkin and K. Loer. Why-because analysis: Formal reasoning about incidents. Technical Report RVS-Bk-01, RVS Group, University of Bielefeld, September 1998.

[LN87] G. Longworth and D. Nicholls. *Structured Systems Analysis and Design Methodology Manual*. NCC Publications, 1987.

[Lon94] J. Lonchamp. An assessment exercise. In *Software Process Modelling and Technology*, Advanced Software Developments, chapter 13, pages 335–356. Research Studies Press, Taunton, England, 1994.

[LSS+99] C.C. Lebow, L.P. Sarsfield, W.L. Stanley, E. Ettedgui, and G. Henning. Safety in the skies: Personnel and parties in ntsb aviation accident investigations. Technical Report MR-1122-ICJ, RAND's Institute for Civil Justice., 1999.

[Lut97] R.R. Lutz. Reuse of a formal model for requirements validation. In C.M. Holloway and K.J. Hayhurst, editors, *4th NASA Langley Formal Methods Workshop*, NASA Conference Publication 3356, NASA Langley Research Centre, Hampton, United States of America, 1997.

[Mai86] T. Maibaum. A logic for the formal requirements specification of real-time/embedded systems. Technical Report FOREST Deliverable Report 3, Imperial College, London, 1986.

[Mai93] T. Maibaum. Temporal reasoning over deontic specifications. In J.-J.Ch. Meyer and R.J. Wieringa, editors, *Deontic logic in computer science: normative system specification*, chapter 7, pages 141–202. John Wiley and Sons Ltd., 1993.

[Mar95] R.B. Marcus. *Modalities: Philosophical Essays*. Oxford University Press, 1995.

[Mau97] D.E. Maurino. Aviation human factors and the safety investigation process. *The International Society of Air Safety Investigators Seminar Proceedings*, 1997.

[McC86] L.T. McCarty. Permissions and obligations: An informal introduction. In A.A. Martino and F.S. Natali, editors, *Automated Analysis of Legal Texts*, pages 307–337. North-Holland, 1986.

[Mel94] P. Mellor. CAD: Computer aided disaster. *High Integrity Systems*, 1(2):101–156, 1994.

[Mey88] J.-J.Ch. Meyer. A different approach to deontic logic: a deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic*, 29:109–136, 1988.

[MF96] W.L. McMullen and B. Findley. Tools database working group report. In *Proceedings of the International Council on Systems Engineering*, volume II, August 1996. Most recent results from the Working Group are available from: `http://www.incose.org/tools/tooltax.html`.

[MLR+97] F. Modugno, N.G. Leveson, J.D. Reese, K. Partridge, and S.D. Sandys. Integrated safety analysis of requirements specification. In *Proceedings of the 3rd International Symposium on Requirements Engineering*, Annapolis, Maryland, January 1997.

[MoP89] Investigation commission concerning the accident which occurred on June 26th 1988 at Mulhouse-Habsheim (68) to the airbus A320, registered F-GKC. Accident report, Ministry of Planning, Housing, Transport and Maritime Affairs, 29 November 1989. Official Translation from the French by the Bureau d'Enquetes Accidents (BEA).

[Mos86] C. Moss. Cut and paste – defining the impure primitives of Prolog. In E.Y. Shapiro, editor, *Proceedings of the Third International Conference on Logic Programming*, volume 225 of *Lecture Notes in Computer Science*, pages 686–694, Imperial College of Science and Technology, London, United Kingdom, July 14–18 1986. Springer-Verlag.

[MP91] Z. Manna and A. Pnueli. On the faithfulness of formal models. In *Mathematical Foundations of Computer Science*, volume 520 of *Lecture Notes in Computer Science*, pages 28–42. Springer-Verlag, 1991.

[Mul82] G.P. Mullery. Acquisition – environment. In G. Goos and J. Hartmanis, editors, *Distributed Systems: Methods and Tools for Specification An Advanced Course*, volume 190 of *Lecture Notes in Computer Science*, chapter 3, pages 45–130. Springer-Verlag, 1982.

[MW93] J.-J.Ch. Meyer and R.J. Wieringa. Deontic logic in computer science: A concise overview. In J.-J.Ch. Meyer and R.J. Wieringa, editors, *Deontic Logic in Computing Science*, chapter 1, pages 3–16. Wiley Professional Computing, New York, 1993.

[Nag88] D.C. Nagel. Human error in aviation operations. In E.L. Wiener and D.C. Nagel, editors, *Human Factors in Aviation*, chapter 9, pages 263–303. Academic Press, London, 1988.

[NE95] S.E. Newstead and J.St.B.T. Evans, editors. *Perspectives on thinking and reasoning: Essays in honour of Peter Wason.* Hove : Erlbaum, 1995.

[Neu95] P.G. Neumann. *Computer-Related Risks.* ACM Press/Addison Wesley, 1995.

[Nor90] D.A. Norman. Human error and the design of computer systems. *Communications of the ACM*, 33(1), January 1990.

[Nut97] D. Nute, editor. *Defeasible Deontic Logic.* Kluwer Academic Publishers, 1997.

[Pau94] L.C. Paulson. *Isabelle: A Generic Theorem Prover.* Springer-Verlag, 1994. Contributions by Tobias Nipkow, Lecture Notes in Computer Science 828.

[PB97] P. Palanque and R. Bastide. Embedding modelling of errors in specifications: costs and benefits. In Chris Johnson, editor, *Workshop on Human Error and Systems Development*, pages 147–159, 1997.

[Pet85] H. Petroski. *To Engineer is Human: The Role of Failure in Successful Design.* Macmillan, London, 1985.

[PFAB86] C. Potts, A.C.W. Finkelstein, M. Aslett, and J. Booth. "Structured Common Sense": A requirements elicitation and formalization method for modal action logic. Technical Report Alvey Initiative FOREST Report R2, Department of Computing, Imperial College of Science, Technology and Medicine, London, 1986.

[Ran97] D.W. Rand. *Concorder.* Les Publications CRM, Montreal, 3 edition, 1997.

[Ras89] J. Rasmussen. Coping safely with complex systems. Technical Report Risø-M-2769, The Risø National Laboratory, Roskilde, Denmark, 1989.

[Ras90] J. Rasmussen. Event analysis and the problem of causality. In J. Rasmussen, B. Brehmer, and J. Leplat, editors, *Distributed Decision Making: Cognitive models for cooperative work*, pages 247–259. John Wiley and Sons Ltd., 1990.

[Rea90] J. Reason. *Human Error.* Cambridge University Press, 1990.

[Rea97] J. Reason. *Managing the Risks of Organizational Accidents.* Ashgate Publishing Ltd., Aldershot, England, 1997.

[RPM+81] J. Rasmussen, O. M. Pedersen, G. Mancini, A. Carnino, and M. Griffon. Classification system for reporting events involving human malfunctions. Technical Report Risø-M-2240, Risø National Laboratory, Roskilde, Denmark, March 1981.

[RTvdT96] J.-F. Raskin, Y.-H. Tan, and L.W.N. van der Torre. How to model normative behavior in Petri nets. In *ModelAge'96: "Formal Models of Agents", Second Workshop of the ModelAge Project*, pages 223–240, Sesimbra, Portugal, 15–17 January 1996. University of Lisbon.

[Rya88] T. Ryan. A task analysis-linked approach for integrating the human factor in reliability assessments of nuclear power plants. *Reliability Engineering and System Safety*, 22:219–234, 1988.

[Sco97] M. Scott. *WordSmith Tools Manual.* Oxford University Press, 1997. Software available at: `http://www4.oup.co.uk/isbn/0-19-459283-9`.

[SF97] G. Spanoudakis and A. Finkelstein. Reconciling requirements: a method for managing interference, inconsistency and conflict. *Annals of Software Engineering (Special Issue on Software Requirements Engineering)*, 3, 1997.

[SJ98] P. Snowdon and C.W. Johnson. The impact of rhetoric on accounts of human 'error' in accident reports. *ISASI Forum*, 31(4), 1998.

[SJ99] P. Snowdon and C.W. Johnson. Results of a preliminary survey into the usability of accident and incident reports. In *People in Control*, Savoy Place, London, United Kingdom, 1999. The Institute of Electrical Engineers.

[SM91] J.W. Senders and N.P. Moray. *Human error: Cause, prediction, and reduction.* Lawrence Erlbaum Associates, Hillsdale, N.J., 1991.

[SOR93] N. Shankar, S. Owre, and J. M. Rushby. *PVS Tutorial.* Computer Science Laboratory, SRI International, Menlo Park, CA, February 1993. Also appears in Tutorial Notes, *Formal Methods Europe '93: Industrial-Strength Formal Methods*, pages 357–406, Odense, Denmark, April 1993.

[SS86] L. Sterling and E. Shapiro. *The Art of Prolog: Advanced Programming Techniques.* MIT Press Series in Logic Programming, 1986.

[SS97a] I. Sommerville and P. Sawyer. *Requirements Engineering: A Good Practice Guide.* John Wiley and Sons, 1997.

[SS97b] M. Spearpoint and M. Shipp. Virtual reality for the channel tunnel fire investigation. *Fire Safety Engineering*, 6(4), 1997.

[Sta93] J. Bowen & V. Stavridou. The industrial take-up of formal methods in safety-critical and other areas: A perspective. In J.C.P. Woodcock and P.G. Larsen, editors, *FME'93: Industrial-Strength Formal Methods, First International Symposium of Formal Methods Europe*, volume 670 of *Lecture Notes in Computer Science*, pages 183–195, Odense, Denmark, April 1993. Formal Methods Europe, Springer-Verlag.

[Str94] G. Struth. Philosophical logics – a survey and a bibliography. Research Report DFKI-RR-94-17, Deutsches Forschungszentrum fur Kunstliche Intelligenz (The German Research Centre for Artificial Intelligence), May 1994.

[Tho92] M.C. Thomas. Published in Software Engineering Notes, pg. 30, April 1992. Quoted in Mellor [Mel94].

[Tho94] M. Thomas. A proof of incorrectness using the LP theorem prover: The editing problem in the Therac-25. *High Integrity Systems*, 1(1):35–49, 1994.

[TO94] M. Thomas and B. Ormsby. On the design of side-stick controllers in fly-by-wire aircraft. *ACM: Applied Computing Review*, 2(1), 1994.

[TR94] B. Toft and S. Reynolds. *Learning from Disasters.* Butterworth-Heinemann Ltd., Oxford, 1994.

[Vau96] D. Vaughan. *The Challenger Launch Decision: Risky Technology, Culture and Deviance at NASA.* University Of Chicago Press, Chicago, 1996.

[vLMvdH97] B. van Linder, J.-J.Ch. Meyer, and W. van der Hoek. Formalising motivational attitudes of agents using the KARO framework. Technical Report UU-CS-1997-03, Utrecht University, 1997.

[vW80] G.H. von Wright. Problems and prospects of deontic logic: A survey. In E. Agazzi, editor, *Modern Logic – A Survey: Historical, Philosophical and Mathematical Aspects of Modern Logic and Its Applications*, chapter 22, pages 399–423. D. Reidel Publishing Company, 1980.

[vW83] G.H. von Wright. Norms, truth and logic. In *Practical Reason: Philosophical Papers*, volume 1. Cornell, 1983.

[Wat97] M.F. Watts. Report on the proposal for a council directive on the registration of persons sailing on board of passenger ships (com(96)0574 - c4-0029/97 - 96/0281(syn)). European Parliament Report A4-0152/97, Committee on Transport and Tourism, European Parliament, 1997.

[WW93] D. Weber-Wulff. Selling formal methods to industry. In J.C.P. Woodcock and P.G. Larsen, editors, *FME'93: Industrial-Strength Formal Methods, Proceedings of the First International Symposium of Formal Methods Europe.*, volume 670 of *Lecture Notes in Computer Science*,

pages 671–678, Odense, Denmark, April 1993. Formal Methods Europe, Springer-Verlag.

Part V

Appendices

Appendix A

Glossary of Terms

It should be noted that these definitions are defined in the context of their application to accident analysis:

Accessibility Relation A relation that, for a given scenario s_i, enumerates the scenarios that are accessible from s_i.

Accident An undesired and unplanned (but not necessarily unexpected) event that results in (at least) a specified level of loss [Lev95].

Accident Report An accident report is the published results of an investigation into an accident performed either internally, by the company involved, or externally, by a governing body.

Action A mechanism through which a system exhibits change [Kho88].

Agent A non-passive object that interacts with and affects its environment. In SCS, agents are defined as 'human users, affected individuals who do not directly use the system or organisation, physical devices, other computer systems and functionally separate components of the system being specified.' [PFAB86]

ALoS Abbreviation for 'Acceptable Level of Safety', used in the Channel Tunnel Fire Report model in Chapter 5.

Attribute A property of an entity.

DAL Deontic Action Logic.

DALEX Executable DAL. A language incorporating a set of operators based on those of EDAL, which is executable using the DALEX interpreter.

DFD Data Flow Diagram

EDAL Extended Deontic Action Logic. A language based on DAL, with a number of extensions to make it better suited for accident analysis.

Entity 'Anything in the domain of discourse that performs or suffers actions, that has properties that change or are of interest, that enters into relationships with other entities that change or are of interest or anything that is a named individual of some significance' [PFAB86].

Error A design flaw or deviation from a desired or intended state [Lev95].

Error of Commission The effect of an agent performing an action they were not permitted to perform.

Error of Omission The effect of an agent failing to perform the expected action.

Failure 'The non-performance or inability of the system or component to perform its intended function for a specified time under specified environmental conditions' [Lev95].

FLOR First Line of Response. Eurotunnel emergency personnel on secondment from Kent Fire Brigade and Pas de Calais fire services [All97].

Formal Language A language with a well-defined syntax and semantics, unambiguously defined in terms of primitive elements.

Hazard A set of conditions that, in conjunction with other conditions, will lead to an accident [Lev95].

Incident An action that 'involves no loss (or only minor loss) but with potential for loss under different circumstance' [Lev95].

Latent Condition Conditions that 'can lie dormant for a time doing no particular time until they interact with local circumstances to defeat the system's defences' [Rea97].

Latent Failure An action that is not permitted, but has no immediate non-normative effect on the behaviour of the system. The outcome of a latent failure is a latent condition.

MAL Modal Action Logic. A logic very closely related to DAL.

Necessitation Rule This rule states that if a property holds, with no extralogical assumptions, then it always holds (i.e., if ϕ is in the theory, $[A, \alpha]\phi$ is also in the theory).

Obliged Action An action that should be performed. If an agent is immediately obliged to perform an action, that action should be the next action it performs.

Permission Structure Also P-Structure. The set of permissions and prohibitions holding in a particular scenario.

Permitted Action An action that may be performed. In (E)DAL, an action that, if performed in a normative scenario by the specified agent, will lead to a normative scenario.

Prohibited Action An action is prohibited if it is not permitted to be performed (by some agent).

Safe A system is safe if it is free from accidents and loss [Lev95].

Safety-critical System A system whose incorrect function (failure) may have very serious consequences such as loss of human life, severe injuries, large-scale environmental damage, or considerable economic penalties [IBN96].

Safety Engineering The discipline of engineering concerned with eliminating hazards or reducing their probability or severity [Lev95].

Scalability The ability of a method or language to represent large, complex systems, such as those commonly described in accident reports, without losing structural clarity.

SCS Structured Common Sense. Requirements elicitation and formalisation technique developed for use with MAL.

Seinsollen Operators State-based deontic operators.

Tunsollen Operators Action-based deontic operators

Appendix B

Case Studies

This appendix presents summaries of the systems and behaviour described in the accident reports used as case studies in this document: 'Inquiry into the fire on heavy goods vehicle shuttle 7539 on 18 November 1996', [All97] and 'The Investigation into the King's Cross Underground Fire', [Fen88]. It also describes the North Anna incident [Dun87, Joh92], which is also referred to in the thesis.

B.1 The Channel Tunnel Fire Report Case Study

The Channel Tunnel is a rail tunnel, owned and operated by Eurotunnel. It consists of three tunnels, running beneath the English Channel, between England and France. The North and South Running Tunnels carry passenger, tourist, Heavy Goods Vehicle (HGV), and freight services, the former generally handling West(England) to East(France) traffic and the latter handling East to West traffic. Between the two lies the Service Tunnel, which is normally used by Eurotunnel maintenance vehicles. There are numerous cross passage doors that connect the service tunnel to the running tunnels.

The Treaty of Canterbury, signed by the governments of the UK and France, required the establishment of two bodies to oversee the construction and operation of the Channel Tunnel. The Intergovernmental Commission was charged with monitoring all aspects of these activities, and the Channel Tunnel Safety Authority was charged with advising and assisting the Intergovernmental Commission with all safety related issues. Members of this safety authority are all experts in the field of safety or rail transport.

Having received procedural and structural plans for the system, the Eurotunnel Management was required to forward these to the Intergovernmental Commission for inspection. The Channel Tunnel Safety Authority expressed concerns about the HGV shuttle designs, but the other designs were approved. The Channel Tunnel Safety Authority later approved the HGV designs after being provided with the results of a number of tests by the Eurotunnel Management.

A week before the fire, an evacuation exercise was performed. The report details that the emergency procedures and the training of the staff were found to be deficient.

On the 18th November 1996, at around 21.48hrs, heavy goods vehicle shuttle 7539 entered the Channel Tunnel from the French side, using the South Running Tunnel. By 21.51 the driver was aware that there was probably a fire on board the train. Policy dictated that the train should continue through the tunnel to the UK terminal where it would be diverted into an emergency siding. However, reports of faults with the props or bridging plates required the train to be brought to a halt or risk derailment. In this scenario, procedures dictated that the driver should decouple the front locomotive and the Amenity Coach, which carried the 31 passengers, and continue out of the tunnel. However, the train lost power due to the tripping of the catenary, most likely due to arcing. The procedure for this situation dictated evacuation of crew and passengers to the Service Tunnel, which has a higher air pressure and thus prevents the incursion of the smoky air from the running tunnel. Unfortunately the thickness of the smoke in the running tunnel made this procedure impossible. The crew and passengers had to wait for it to thin before they could evacuate. One of the factors that delayed its thinning was the incorrect configuration of the ventilation system. The majority of passengers were then evacuated in a train that had been stopped in the North running tunnel. The others were treated in a Service Tunnel ambulance vehicle and later evacuated in Service Tunnel Transport vehicles.

The Emergency Services consisted of the French and UK Eurotunnel first lines of response (fire-fighting personnel seconded to Eurotunnel from the respective National fire services), and the emergency services of the two countries. Communications between the various centres of each country and with the fire fighters at the fire location were found by the inquiry to have often been inadequate.

B.2 The London King's Cross Underground Fire

On the 18th November, 1987, a fire broke out on escalator 4 of the Piccadilly line in King's Cross Underground Station, London. It left 31 people dead and many seriously injured.

Despite smoking having been banned within the Underground for a number of years, poor policing of this policy had led to passengers continuing to smoke freely within the premises. On this occasion, a lit match was dropped on the escalator 4 where, due to the crabbing movement of the steps, it slipped between the skirting board and the step and landed on the running tracks of the escalator. Years of incomplete maintenance had resulted in an accumulated layer of grease and detritus that the match ignited. Once lit, the fire spread upwards to the escalator wall.

Mr Squire was the first person to report the fire, to Booking Clerk Newman, at around 19:29. However, Mr Squire incorrectly located the fire and thus Relief Station Inspector Hayes, who was informed of the fire by Booking Clerk Newman, was delayed in getting to the actual location of the fire. Once there he tried to tackle the blaze from beneath

with an extinguisher, but could not get near enough. Staff, police and other passengers took action such as stopping and blocking the escalators, redirecting passengers up the Victoria Line escalators, informing the London Underground management, calling the Fire Brigade, ordering trains not to stop and finally evacuating the remainder of the staff. The Fire Brigade had arrived and were in the process of initial inspection of the fire when the 'flashover' occurred - a jet of flames shot up the shaft of the escalator accompanied by intense heat and thick black smoke. Most of the deaths and injuries occurred at this time. In the quarter hour since the fire had first been reported, no water had been applied to it.

B.3 The North Anna Incident

After the Three Mile Island incident in 1979, the United States Nuclear Regulatory Commission adopted a policy of minimal operator intervention in the workings of the reactors, which affected the regulations that followed. At the North Anna nuclear reactor, there had been changes in the generating processes. Following a reactor scram, there were dangerous temperature profiles. The operators were faced with the choice of following the prescribed behaviour of the Nuclear Regulatory Commission or breaking these regulations and preventing the endangerment of the plant. Fortunately, the latter was chosen.

Appendix C

EDAL Syntax

C.1 Language

C.1.1 Syntactic Categories

These syntactic categories are identical to those of DAL [Kho88], with the exception of the operator names and the absence of the agent sort.

Sorts A non-empty finite collection of sorts $\mathcal{S}$. $\mathcal{S} = Act \cup S(Act) \cup AAP \cup AS(Act) \cup \{s_1, s_2, \ldots\}$, where Act denotes the predefined action sort, $S(Act)$ denotes the predefined sort of action sequences, and AAP denotes the predefined sort of agent-action pairs, and $AS(Act)$ denotes the predefined sort of agent-action pair sequences. $\{s_1, s_2, \ldots\}$ denote the other sorts of entity to be modelled.

Constant Symbols For each sort $s \in \mathcal{S}$, there exists a set of constant symbols of sort s. In addition, there is the predefined normativity constant, η.

Action Names For each $n > 0$, and each n-tuple, $\langle s_1, \ldots, s_n \rangle$, such that $s_1 \in \mathcal{S}, \ldots, s_{n-1} \in \mathcal{S}$ and $s_n \in Act$, there is a set of n-place action names each of which is of sort $\langle s_1, \ldots, s_n \rangle$.

Predicate Symbols :

For each n>0, and each tuple, $\langle s_1, \ldots, s_n \rangle$, such that $s_1 \in \mathcal{S}, \ldots, s_n \in \mathcal{S}$, there is a set of n-place predicate symbols each of which is of sort $\langle s_1, \ldots, s_n \rangle$. Predefined predicate symbols are: IP, IO, O, P, OS, PS.

Function Symbols :

For each n>0 and each n+1 tuple, $\langle s_1, \ldots, s_n, s_{n+1} \rangle$ such that $s_1 \in \mathcal{S} \ldots s_{n+1} \in \mathcal{S}$, there is a set of n-place function symbols each of which is said to be of sort $\langle s_1, \ldots, s_{n+1} \rangle$. Predefined function symbols are: ;, ||, +, <>, and ,.

Variables For each sort $s \in \mathcal{S}$, there are a set of distinct variable symbols.

Quantifiers For each sort $s \in \mathcal{S}$, the quantifiers $\forall_s$ and $\exists_s$ exist.

Logical Operators :

First-Order $\neg, \leftrightarrow, \rightarrow, \wedge, \vee$.
Modal $[\_,\_]\_$, $\langle\_,\_\rangle\_$
(Seinsollen) Deontic $Per(\_,\_)$, $Obl(\_,\_)$, $Sper(\_,\_)$, $Sobl(\_,\_)$.

Equality Symbols For each sort $s \in \mathcal{S}$, there is the infix equality operator $=_s$ of sort $\langle s, s\rangle$.

Punctuation The symbols (,), and ,.

C.1.2 Formation Rules

These formation rules are as given in Chapter 2

Terms :

- For each sort $s \in \mathcal{S}$, a variable or constant of sort s is a term.
- If $t_1 \ldots t_n$ are terms of sorts $s_1 \ldots s_n$ respectively and $s_1 \ldots s_n$ are all taken from $\mathcal{S}$ and f is a function symbol of sort $\langle s_1 \ldots s_n, s_{n+1}\rangle$ then $f(t_1 \ldots t_n)$ is a term of sort s_{n+1}.
- If $t_1 \ldots t_n$ are terms of sorts $s_1 \ldots s_n$ respectively and $s_1 \ldots s_n$ are all taken from $\mathcal{S}$ and a is an action symbol of sort $\langle s_1 \ldots s_n\rangle$ then $a(t_1 \ldots t_n)$ is a term of sort *Act*.
- Nothing else is a term.

Atoms (Atomic Formulae):

- If $t_1 \ldots t_n$ are terms of sorts and $s_1 \ldots s_n$ respectively and $s_1 \ldots s_n$ are all taken from $\mathcal{S}$ and p is a predicate symbol of sort $\langle s_1 \ldots s_n\rangle$ then $p(t_1 \ldots t_n)$ is an atom.
- For each sort $s \in \mathcal{S}$, given two terms of s, t_1 and t_2, $t_1 =_s t_2$ is an atom, provided $=_s$ is an equality symbol of the language.
- Nothing else is an atom.

Formulae :

- An atom is a formula.
- The logical constant, η, is a formula.
- If ϕ is a formula, so is $\neg\phi$.

- If ϕ and φ are formulae then $(\phi \vee \varphi)$, $(\phi \wedge \varphi)$, $(\phi \rightarrow \varphi)$, and $(\phi \leftrightarrow \varphi)$ are also formulae.
- If α is a term of sort Act, A is a term of sort Agt, and ϕ is a formula, then $[A, \alpha]\phi$ is also a formula.
- If ϕ is a formula and A is a term of sort Agt then $Per(A, \phi)$, $Obl(A, \phi)$, $Sper(A, \phi)$, $Sobl(A, \phi)$ are formulae.
- For each sort $s \in \mathcal{S}$, if x is a variable of sort s and ϕ is a formula, then $\forall_s x.\phi$ and $\exists_s x.\phi$ are formulae.
- If σ is of sort $S(Act)$, A is a term of sort Agt and ϕ is a formula, the $[A, \sigma]\phi$ is a formula.
- Nothing else is a formula

C.2 Axioms

C.2.1 FOL Axioms

With the exception of EQ, the equality axioms, such as reflexivity, symmetry and transitivity, are omitted, but are as usual. If ϕ, ψ and τ are formulae, α, β are terms of sort Act, A is a term of sort Agt, σ is a term of sort $S(Act)$ then the following are axioms:

(E1) $\phi \rightarrow (\psi \rightarrow \phi)$
(E2) $(\tau \rightarrow (\phi \rightarrow \psi)) \rightarrow ((\tau \rightarrow \phi) \rightarrow (\tau \rightarrow \psi))$
(E3) $(\neg\phi \rightarrow \neg\psi) \rightarrow ((\neg\phi \rightarrow \psi) \rightarrow \phi)$
(E4) $\forall x \phi(x) \rightarrow \psi(t)$ where t is free for x in ϕ.
(E5) $(\forall x(\phi \rightarrow \psi)) \rightarrow (\phi \rightarrow \forall x \psi)$
where x is not free in ϕ
(EQ) $\forall x, y((\phi(x) \wedge x = y) \rightarrow \phi(y))$
where y is not within the scope of a modal operator

C.2.2 Modal Operator Axioms

(E6) $[A, \alpha](TRUE)$
(E7) $([A, \alpha](\phi \rightarrow \psi)) \leftrightarrow (([A, \alpha]\phi) \rightarrow ([A, \alpha]\psi))$
(E8) $([A, \alpha]\neg\phi) \leftrightarrow (\neg[A, \alpha]\phi)$
(E9) $\forall x([A, \alpha]\phi) \leftrightarrow ([A, \alpha]\forall x \phi)$
where x is not free in A or α
(E10) $\exists x([A, \alpha]\phi) \leftrightarrow ([A, \alpha]\exists x \phi)$
where x is not free in A or α
(E11) $(([A, \alpha]\phi) \vee ([A, \alpha]\psi)) \leftrightarrow ([A, \alpha]\phi \vee \psi)$
(E12) $(([A, \alpha]\phi) \wedge ([A, \alpha]\psi)) \leftrightarrow ([A, \alpha]\phi \wedge \psi)$

C.2.3 Deontic Axioms

(E13) $\eta \wedge IP(A,\alpha) \rightarrow [A,\alpha]\eta$
(E14) $\neg[A,\alpha]\eta \rightarrow [A,\alpha]\neg\eta$
(E15) $IO(A,\alpha) \rightarrow IP(A,\alpha)$
(E16) $IO(A,\alpha) \wedge \neg[A,\alpha]IO(A,\alpha) \rightarrow [A,\alpha]\neg IO(A,\alpha)$
(E17) $P(A,\alpha) \wedge (IO(A,\beta) \rightarrow (\alpha = \beta)) \rightarrow IP(A,\alpha)$
(E18) $\neg P(A,\alpha) \wedge \neg IO(A,\alpha) \rightarrow \neg IP(A,\alpha)$
(E19) $O(A,\alpha) \rightarrow \exists\sigma[A,\sigma]\neg O(A,\alpha) \wedge PS(A,\sigma)$
(E20) $O(A,\alpha) \wedge \neg[A,\alpha]O(A,\alpha) \rightarrow [A,\alpha]\neg O(A,\alpha)$

C.2.4 Latent Failures

(E21) $\eta \rightarrow$
$(\neg IP(A,\alpha) \rightarrow [A,\alpha](latentc(A,\alpha) \rightarrow \eta))$
(E22) $(\alpha \neq \beta) \wedge IO(A,\alpha) \wedge \neg latento(A,\alpha) \rightarrow \neg IP(A,\beta)$
(E23) $(\alpha \neq \beta) \wedge IO(A,\alpha) \wedge latento(A,\alpha) \wedge P(A,\beta) \rightarrow IP(A,\beta)$
(E24) $(\alpha \neq \beta) \wedge IO(A,\alpha) \wedge latento(A,\alpha) \wedge latentc(A,\beta)) \rightarrow IP(A,\beta)$
(E25) $(\alpha \neq \beta) \wedge IO(A,\alpha) \wedge latento(A,\alpha) \wedge \neg(P(A,\beta) \vee latentc(A,\beta)) \rightarrow$
$\neg IP(A,\beta)$

C.2.5 Combining, Sequencing and Seinsollen Axioms

(E26) $[A,<\alpha>]\phi \leftrightarrow [A,\alpha]\phi$
(E27) $[A,<\sigma,\alpha>]\phi \leftrightarrow [A,\sigma][A,\alpha]\phi$
(E28) $PS(A,<\alpha>) \leftrightarrow P(A,\alpha)$
(E29) $PS(A,<\sigma,\alpha>) \leftrightarrow (PS(\sigma) \wedge [\![A,\sigma]\!]IP(A,\alpha))$
(E30) $OS(A,<\alpha>) \leftrightarrow O(A,\alpha)$
(E31) $OS(A,<\sigma,\alpha>) \leftrightarrow (OS(A,\sigma) \wedge [\![A,\sigma]\!]IO(A,\alpha))$
(E32) $P(A,\alpha;\beta) \rightarrow ([A,\alpha;\beta]\phi \leftrightarrow [A,\alpha][A,\beta]\phi)$
(E33) $P(A,\alpha \parallel \beta) \rightarrow (([A,\alpha]\phi \wedge [A,\beta]\psi) \rightarrow ([A,\alpha \parallel \beta]\phi \wedge \psi))$
(E34) $\alpha \parallel \beta =_{Act} \beta \parallel \alpha$
(E35) $P(A,\alpha+\beta) \rightarrow (([A,\alpha]\phi \wedge [A,\beta]\psi) \rightarrow ([A,\alpha+\beta]\phi \vee \psi))$
(E36) $([A,\alpha]\phi \wedge [A,\beta]\psi \wedge \neg[A,\alpha]\psi \wedge \neg[A,\beta]\phi) \rightarrow (\neg[A,\alpha+\beta]\phi \wedge \psi)$
(E37) $\alpha+\beta =_{Act} \beta+\alpha$
(E38) $Per(A,\phi) \leftrightarrow \exists\alpha.[A,\alpha]\phi \wedge IP(A,\alpha)$
(E39) $Obl(A,\phi) \leftrightarrow \exists\alpha.[A,\alpha]\phi \wedge IO(A,\alpha)$
(E40) $Sper(A,\phi) \leftrightarrow \exists\sigma.[\![A,\sigma]\!]\phi \wedge PS(A,\sigma)$
(E41) $Sobl(A,\phi) \leftrightarrow \exists\sigma.[\![A,\sigma]\!]\phi \wedge OS(A,\sigma)$

C.3 Proof Theory

The provability relation, denoted by the symbol, $\vdash$, is of sort: P(Form(L)) X Form(L), where L is the language of the logic, Form(L) is a formula in the language, and P(Form(L)) is the powerset of formula in the language. The provability relation symbolising provability from a specific scenario is given as $\vdash_s$, where s is the theory presentation $< L, A >$ of the theory representing the scenario $t(s)$.

The inference rules of EDAL are:

$$\frac{\vdash_s \phi}{\vdash_s \forall x \phi} \tag{C.1}$$

$$\frac{\vdash_s \phi \quad \vdash_s \phi \rightarrow \psi}{\vdash_s \psi} \tag{C.2}$$

$$\frac{\vdash_s \phi}{\underline{\vdash}_s \neg \phi} \tag{C.3}$$

$$\frac{\vdash_s \phi \ \underline{\vdash}_s [A, \alpha] \neg \phi}{\vdash_s [A, \alpha] \phi} \tag{C.4}$$

Appendix D

EDAL Semantics

An EDAL semantic model for an EDAL *specification* is a set of scenarios and some scenario changing functions. To define what is meant by a semantic model and a specification first requires the definition of theories and theory presentations.

Definition 10 (Theory Presentation) *A theory presentation TP is a pair $\langle L_{TP}, A_{TP} \rangle$. L_{TP} is a language description (i.e., the sorts and arities of operations) and A_{TP} is a set of axioms (in the language L_{TP}).*

EDAL specifications and scenarios are theory presentations. For example, the specification *sp* is the following:

$$L_{sp} = \begin{cases} Sorts & = & Nat, Act, Agt, Book \\ Constants & = & \\ & & Tom : Agt \\ & & 0 : Nat \\ & & b1, b2 : Book \\ Variables & = & \\ & & p, q : Agt \\ & & x : Nat \\ & & b : Book \\ Function\_Symbols & = & \_ + \_ : Nat \times Nat \rightarrow Nat \\ Action\_Names & = & \\ & & returnbook : Book \rightarrow Act \\ & & getbook : Book \rightarrow Act \end{cases}$$

$$A_{sp} = \begin{cases} x + 0 = x \\ [p, getbook(b)]O(p, returnbook(b)) \end{cases}$$

In subsequent examples, we omit L_{TP} (for a given TP) when it can be deduced from A_{TP}.

Definition 11 (Specification Theory) *Given an EDAL specification sp, $SpTH(sp)$ is inductively defined by:*

$$1\ \ (A_{sp} \vdash \phi) \rightarrow \phi \in SpTH(sp)$$

$$2\ \ (\phi \in SpTH(sp) \wedge A : Agt \in L_{sp} \wedge \alpha : Act \in L_{sp}) \rightarrow [A, \alpha]\phi \in SpTH(sp)$$

$\vdash$ denotes provability using the specification axioms (i.e., A_{sp}) and the axioms of FOL and EDAL.

Note that this definition differs from the DAL definition, as it explicitly includes the necessitation rule for specifications.

For example, the specification theory of the specification *sp* from the previous example is:

$$SpTH(sp) = \left\{ \begin{array}{l} x + 0 = x \\ 1 + 0 = 1 \\ 2 + 0 = 2 \\ 3 + 0 = 3 \\ 4 + 0 = 4 \\ 5 + 0 = 5 \\ \ldots \\ [p, getbook(b)]O(p, returnbook(b)) \\ [p, getbook(b)][q, getbook(b)]O(q, returnbook(b)) \\ [Tom, getbook(b1)]x + 0 = x \\ \ldots \end{array} \right.$$

Definition 12 (Scenario Theory) *Given a specification sp and a scenario s, where s and sp share the same language, $TH_{sp}(s)$ is the set of the formulae that are provable using the scenario axioms (i.e., A_s), the FOL, SFOAL, and EDAL axioms and the formulae in the specification theory. Formally:*

$$TH_{sp}(s) = \{\phi \mid A_s \cup SpTH(sp) \vdash \phi\}$$

Given the specification *sp* from the previous examples, a scenario *s* and its scenario theory $TH_{sp}(s)$ could be given as follows:

$$s = \left\{ \begin{array}{l} O(Tom, returnbook(b1)) \\ O(Tom, returnbook(b1)) \\ O(Tom, returnbook(b1)) \rightarrow P(Tom, returnbook(b1)) \end{array} \right.$$

$$TH_{sp}(s) = SpTH(sp) \cup \left\{ \begin{array}{l} O(Tom, returnbook(b1)) \\ O(Tom, returnbook(b2)) \\ O(Tom, returnbook(b1)) \rightarrow P(Tom, returnbook(b1)) \\ P(Tom, returnbook(b1)) \end{array} \right.$$

In subsequent examples we omit sp from $TH_{sp}(TP)$ as the scenario theories we discuss always use the same specification theory. We henceforth use the term *theory* as shorthand for scenario theory.

A theory that includes at least the same properties as another, is said to be an extension of the latter.

Definition 13 (Theory Extension) *Given two scenarios s and s' (where $L_s = L_{s'}$), $TH(s')$ is said to be a theory extension of $TH(s)$ iff at least the same properties are derivable in s' as in s, written $s \subseteq s'$. Formally:*

$$\forall \phi.\ If\ \phi \in TH(s) Then\ \phi \in TH(s')$$

Note that this implies that, for a given specification sp, all theories are theory extensions of the scenario theory of the specification, i.e., $sp \subseteq s$.

Definition 14 (Consistency of Scenarios) *Given a scenario s and a formula ϕ, s is consistent iff it is not the case that both ϕ and $\neg\phi$ hold in $TH(s)$.*

An EDAL model structure is the behavioural definition of the specification.

Definition 15 (EDAL Model Structure) *A model structure for an EDAL specification sp is given by the pair $\langle \mathcal{S}, \mathcal{F} \rangle$ where:*

- $\mathcal{S}$ *is a non empty set of potential scenarios such that*
  $\mathcal{S} \subseteq \{s : sp \sqsubseteq s\ \&\ s\ is\ consistent\}$
- $\mathcal{F}$ *is a collection of typed (total) functions indexed by action names in sp.*
  $\mathcal{F}_{t(\alpha)} : \mathcal{S} \rightarrow \mathcal{S}$
  where $t(\alpha)$ is the type of the action name.

An EDAL model is an EDAL model structure that is constrained by a number of properties.

Definition 16 (EDAL Model) *An EDAL model structure $\mathcal{M}$ is a model for an EDAL specification sp iff the following trans-scenario property holds in $\mathcal{M}$:*

If the theory $TH(s)$ contains an action description stating that the property ϕ holds as after agent A performs action α, then ϕ holds in the theory accessed from $TH(s)$ by the performance of α by A.

$$\forall s \in S, \forall A \in Agt. \forall \alpha \in Act.$$
$$If\ [A,\alpha]\phi \in TH(s)\ Then\ \phi \in TH(f_{<A,\alpha>}(s))$$

Appendix E

Channel Tunnel Fire Model

Agent Sorts:

CFDS	%Complete Fire Detection System
CTSA	%Channel Tunnel Safety Authority
D_C	%Designers and Constructors of Channel Tunnel System
E_MAN	%Eurotunnel Management
EMS	%Engineering Management System
GM	%Government Ministers
IGC	%Inter-Governmental Commission
RTMS	%Rail Traffic Management System
TECH_EX	%Technical Experts
TESTER	%Testing agents
UK_F	%UK and French Governments

Agent Constants:

cfds: CFDS
ctsa: CTSA
dc: D_C
e_man: E_MAN
ems: EMS
gm: GM
igc: IGC
rtms: RTMS
tech_ex: TECH_EX
tester: TESTER
ukf: UK_F

Other Sorts:

DESIGN

DESIGN_CONCERN
DESIGN_SAFETY
DESIGN_STATUS
POLICY
TREATY

Other Constants:
hGVDesign, genDesign: DESIGN
none, unaddressed, addressed: DESIGN_CONCERN
unavailable, none, well_designed_reliable_equipement,
clear_effective_procedures, both: DESIGN_SAFETY
unfinished , unscrutinised, approved, unapproved: DESIGN_STATUS
minSepPolicy, dgPolicy: POLICY
%Minimum Separation Policy, Dangerous Goods Policy
cantTreaty: TREATY
%Treaty of Canterbury, signed by two governments

Predicates:
signed: TREATY
established: CTSA
established: IGC
arrangedScrutiny: DESIGN
ctsaApproval: DESIGN
announcedConcern: DESIGN
sentAssess: DESIGN
permit: DESIGN
testCfds
testRtms
testFireDev
testProc
testVent
implemented: POLICY
testResults: TEST_RESULTS
permitApplied: DESIGN
status: DESIGN X DESIGN_STATUS
safety: DESIGN X DESIGN_SAFETY
testedCfds
testedVent
testedProc
testedRtms
testedFireDev
testResults %The combination of outcomes of the preceding 5 tests.
concern: DESIGN X DESIGN_CONCERN
furtherEvents
deficientProcedures
improperImplementation

safetyAdvice: CTSA X IGC
igcSupervise
incident: INCIDENT
trained: E_STAFF

Actions:
(uk_f) Sign: TREATY
(gm) Establish: IGC
(gm) Establish: CTSA
(ctsa) Arrange_Scrutiny: DESIGN
(ctsa) Express_Concerns: DESIGN
(ctsa) Approve_Safety: DESIGN
(igc) Safety_Assess: DESIGN
(igc) Grant_Permit: DESIGN
(e_man) Request: AGENT X ACTION
(e_man) Implement_Policy: POLICY
(e_man) Train: E_STAFF
(e_man) Apply_Permit: DESIGN
(e_man) Test_Ex
(tech_ex) Scrutinise_Safety: DESIGN
(dc) Produce_Design: DESIGN X DESIGN_SAFETY
(cfds) Reaction_Test
%Test the Reaction Time of Fire Detection
(ems) Vent_Test
%Test Ventilation System
(tester) Em_Proc_Test
%Test Emergency Procedures
(tester) HGV_RoF_Dev_Test
%Test Rate of Fire Development in HGV Train
(rtms) Separation_Test
%Test Separation Distance Between Trains

Variables:
α,β,γ: ACT
CF: CFDS
CT: CTSA
D: DESIGN
DC: D_C
DCON, DCON2: DESIGN_CONCERN
DS, DS2: DESIGN_STATUS
E: E_MAN
ES: E_STAFF
EM: EMS
G: GM
I: IGC
P: POLICY

RT: RTMS
S, S2: DESIGN_SAFETY
T: TREATY
TES: TESTER
TR: TEST_RESULTS
TX: TECH_EX
UF: UK_F

$\%----------$
%Generic Action Descriptions
$\%----------$

%UK&France
$[UF, Sign(T)]signed(T)$

%Government Ministers
$[G, Establish(CT)]established(CT)$
$[G, Establish(I)]established(I)$

%CTSA
$[CT, Arrange\_Scrutiny(D)]arrangedScrutiny(D)$
$[CT, Approve\_Safety(D)]ctsaApproval(D)$
$[CT, Express\_Concerns(D)]announcedConcern(D)$

%Intergovernmental Commission
$[I, Safety\_Assess(D)]sentAssess(D)$
$[I, Grant\_Permit(D)]permit(D)$

%Eurotunnel Management
$[E, Implement(P)]implemented(P)$
$[E, Train(ES)]trained(ES)$
$[E, Apply\_Permit(D)]permitApplied(D)$

%Designers and Constructors
$[DC, Produce\_Design(D, S)]status(D, unexamined) \wedge safety(D, S)$

%Complete Fire Detection System

$[CF, Reaction\_Test]testedCfds \wedge \neg testCfds$

$\%Engineering\ Management\ System$
$[EM, Test\_Vent]testedVent \wedge \neg testVent$

$\%Tester$
$[TES, Em\_Proc\_Test]testedProc \wedge \neg testProc$
$[TES, HGV\_RoF\_Dev\_Test]testedFireDev \wedge \neg testFireDev$

$\%Rail\ Traffic\ Management\ System$
$[RT, Separation\_Test]testedRtms \wedge \neg testRtms$

$\% * * * * * * * * * * * * * * *$
$\%Report\ Specific\ Action\ Descriptions$
$\% * * * * * * * * * * * * * * *$

$[e\_man, Present\_Evidence(hgv)]$
$\quad status(hgv, approved) \wedge concern(hgv, addressed)$
$[e\_man, Test\_Ex]furtherEvents$
$[e\_man, Test\_Ex]deficientProcedures \wedge impropcrImplementation$
$[e\_man, Request(cfds, Reaction\_Test)]testCfds$
$[e\_man, Request(rtms, Separation\_Test)]testRtms$
$[e\_man, Request(tester, Hgv\_RoF\_Dev_{T}est)]testFireDev$
$[e\_man, Request(tester, Test\_Em\_Proc)]testProc$
$[e\_man, Request(ems, Test\_Vent)]testVent$

$[tech\_ex, Examine\_Safety(gen)]$
$\quad status(gen, approved) \wedge concern(gen, none)$
$[tech\_ex, Examine\_Safety(hgv)]$
$\quad status(D, unapproved) \wedge concern(hgv, unaddressed)$

$\% * * * * * * * * * * * * * * *$
$\%Generic\ FOL\ Axioms$
$\% * * * * * * * * * * * * * * *$

$deficientProcedures \rightarrow$
$\quad \neg\forall D : DESIGN.(safety(D, both) \vee safety(D, unavailable))$

$status(D, unfinished) \rightarrow safety(D, unavailable)$
$status(D, DS) \wedge DS \neq unfinished \rightarrow safety(D, S) \wedge S \neq unavailable$
$status(D, DS) \rightarrow ((DS \neq DS2) \rightarrow \neg status(D, DS2))$
$safety(D, S) \rightarrow ((S \neq S2) \rightarrow \neg safety(D, S2))$
$concern(D, DCON) \rightarrow ((DCON \neq DCON2) \rightarrow \neg concern(D, DCON2))$

$\%**********$
$\%Report\ Specific\ FOL\ Axioms$
$\%**********$

$true \rightarrow \neg trained(e\_staff)$

$established(ctsa) \rightarrow safetyAdvice(ctsa, igc)$
$established(igc) \rightarrow igcSupervise$

$status(gen, examined) \rightarrow concern(gen, none)$
$status(hgv, examined) \wedge \neg(concern(hgv, addressed)) \rightarrow$
$concern(hgv, unaddressed)$

$testedCfds \wedge testedRtms \wedge testedVent \wedge testedFireDev \wedge testedProc \rightarrow$
$testResults$

$igcSupervise \wedge incident(fire) \wedge \neg(investigate(CR, fireIncReport)) \rightarrow$
$reportStatus(fireIncReport, unwritten)$

$\%**********$
$\%Generic\ Deontic\ Axioms$
$\%**********$

$\neg(signed(T)) \rightarrow P(UF, Sign(T))$
$signed(T) \rightarrow \neg(P(UF, Sign(T)))$

$safety(D, unavailable) \rightarrow$
$\forall S : SAFETY.\neg((S = both) \vee (P(Produce\_Design(D, S))$

$status(D, unexamined) \rightarrow$

$P(E, Apply\_Permit(D)) \wedge O(E, Apply\_Permit(D))$

$safetyAdvise(CT, I) \wedge sentAssess(D) \wedge \neg arrangedScrutiny(D) \rightarrow$
$IO(CT, Arrange\_Scrutiny(D))$
$status(D, approved) \rightarrow P(CT, Approve\_Safety(D))$

$igcSupervise \wedge permitApplied(D) \wedge \neg(sentAssess(D)) \rightarrow$
$IO(igc, Safety\_Assess(D))$
$igcSupervise \wedge ctsaApproval(D) \rightarrow P(igc, Grant\_Permit(D))$

$concern(D, unaddressed) \wedge \neg(announcedConcern(D)) \rightarrow$
$IO(CT, Express\_Concerns(D))$

$arrangedScrutiny(D) \wedge status(D, unexamined) \rightarrow$
$IO(TX, Examine\_Safety(D))$

$status(D, unfinished) \rightarrow P(DC, Produce\_Design(D, both))$
$status(D, unfinished) \rightarrow \neg P(DC, Produce\_Design(D, S)) \wedge \neg(S = both)$
$\neg status(D, unfinished) \rightarrow \neg P(DC, Produce\_Design(D, S))$

$testCfds \rightarrow IO(CF, Reaction\_Test)$
$testProc \rightarrow IO(TES, Em\_Proc\_Test)$
$testFireDev \wedge testedProc \rightarrow IO(TES, Hgv\_RoF\_Dev\_Test)$
$testVent \rightarrow IO(EM, Test\_Vent)$
$testRtms \rightarrow IO(RT, Separation\_Test)$

%**********
%*Report Specific Deontic Axioms*
%**********

$signed(cantTreaty) \wedge \neg established(ctsa) \rightarrow IO(gm, Establish(ctsa))$
$signed(cantTreaty) \wedge established(ctsa) \wedge \neg(established(igc)) \rightarrow$
$IO(gm, Establish(igc))$

$announcedConcern(hgv) \wedge testResults \rightarrow$
$P(e\_man, Present\_Evidence(hgv))$
$true \rightarrow$

$(P(e\_man, Request(cfds, Reaction\_Test) \wedge$
$P(e\_man, Request(rtms, Separation\_Test) \wedge$
$P(e\_man, Request(tester, HGV\_RoF\_Dev\_Test) \wedge$
$P(e\_man, Request(ems, Vent\_Test) \wedge$
$P(e\_man, Request(tester, Em\_Proc\_Test))$

$igcSupervise \wedge permit(hgv) \wedge \neg(implemented(minSepPolicy)) \rightarrow$
$IO(e\_man, Implement(minSepPolicy))$
$igcSupervise \wedge permit(hgv) \wedge (implemented(minSepPolicy)) \wedge$
$\neg(implemented(dgPolicy)) \rightarrow$
$IO(e\_man, Implement(dgPolicy))$
$igcSupervise \wedge permit(hgv) \wedge (implemented(dgPolicy)) \wedge$
$\neg(trained(e\_staff)) \rightarrow$
$IO(e\_man, Train(e\_staff))$
$igcSupervise \wedge permit(gen) \wedge \neg(trained(e\_staff)) \rightarrow$
$IO(e\_man, Train(e\_staff)$
$igcSupervise \wedge permit(gen) \wedge permit(hgv) \rightarrow$
$P(e\_man, Test\_Ex) \wedge O(e\_man, Test\_Ex)$
$igcSupervise \wedge incident(fire) \wedge \neg(incidentReport(fireIncReport)) \rightarrow$
$IO(e\_man, Report\_Incident(fireIncReport))$

$\%**********$
$\%Latent\ Failures$
$\%**********$

$\%Prescription\ error$
$\neg(concern(D, unaddressed)) \wedge$
$safety(D, well\_designed\_reliable\_equipment) \rightarrow$
$latentc(ctsa, Approve\_Safety(D))$

$\%Qualitative\ error$
$latentc(dc, Produce\_Design(D, well\_designed\_reliable\_equipment))$

$\%Error\ of\ omission$
$latento(e\_man, Train(e\_staff))$

$\%**********$

$\%$'*Acceptable Level of Safety Axioms*

$\%$ * * * * * * * * * * * * * * *

$dcAlos \rightarrow$

$\forall D : DESIGN.(safety(D, both) \vee safety(D, unavailable))$

$ctsaAlos \rightarrow$

$\forall D : DESIGN.(ctsaApproval(D) \rightarrow$

$(\neg(concern(D, unaddressed)) \wedge safety(D, both)))$

$emanAlos \rightarrow$

$(permit(hgv) \rightarrow$

$[e\_man, \alpha][e\_man, \beta][e\_man, \gamma]$

$trained(e\_staff) \wedge implemented(minSepValue) \wedge$

$implemented(dgPolicy))$

$\wedge$

$(permit(gen) \rightarrow [e\_man, \alpha]trained(e\_staff))$

Appendix F

Channel Tunnel Fire Agent Hierarchy

Governmental Agents
- Intergovernmental Agents
  - **Intergovernmental Commission**
  - **Chairman of CTSA**
  - **UK & France**
  - **Government Ministers**
  - **CTSA**
  - **Co-Rapporteurs**
  - **Intergovernmental Commission**
- French Government
  - **Prefet**

Eurotunnel
- **Eurotunnel Management**
- **All Eurotunnel Staff**
- Eurotunnel Staff
  - **AllDrivers**
  - Drivers
    - **Train 6523 Driver**
    - **Train 4899 Driver**
    - **Evacuation Train/ Train 6518 Driver**
    - **Incident Train Driver**
    - **Train 752 Driver**
    - **Train 7533 Driver**
    - **Train 7532 Driver**
    - **Train 6527 Driver**
    - **Train 9059 Driver**

- Chef de Trains
  - **Incident Train Chef de Train**
  - **Evacuation Train Chef de Train**
- Stewards
  - **Incident Train Steward**
- STTS Vehicle Drivers
  - **STTS Ambulance Driver**
  - **STTS Vehicle Driver**
- Terminal Staff
  - French
    - **Dangerous Goods Officer**
    - **Loaders**
- Emergency Response Teams
  - **French FLOR**
  - **UK FLOR**
- French Security
  - **Security in Building**
  - **Dog Handler**
  - **Security at Tunnel Entrance**
  - **Security Supervisor**

Eurotunnel Systems/Centres

- On Train
  - In-Cabin Display
    - **Incident Train In Cabin Display**
    - **Evacuation Train In Cabin Display**
    - **Train 6523 In Cabin Display**
  - On Train Fire Detection System (FDS)
    - **Incident Train On-Train FDS**
    - **Evacuation Train On-Train FDS**
  - Locomotive FDS
    - **Incident Train Locomotive FDS**
  - ATP
    - **Evacuation Train ATP**
    - **Train 6523 ATP**
- UK Terminal
  - **UK Fire Equipment Management Centre**
  - **UK Incident Control Centre**
  - **UK Terminal Control Centre**
  - Rail Control Centre
    - **Rail Control Centre**
    - **Rail Traffic Management System**
    - **Engineering Management System**
- In Tunnel
  - **In Tunnel FDS**

French Terminal

- **French Incident Control Centre**

- **French Fire Equipment Management Centre**
- **French TCC**

Emergency Services
- **Chief Constable of Kent**
- **Chief Fire Officer of Kent**
- **Chief Exec of Kent Ambulance**
- **Members of the UK & French Emergency Services**
- **Medical Team**
- **Transfer Drivers**

External/Other Agents
- Non-Governmental Agents
  - **Technical Experts**
  - **Testers**
  - **Designers & Constructors**
  - **Passengers**
  - **Railway Networks**
  - **Striking Eurotunnel Staff**
- **Fire**
- **Environment**

Appendix G

The DALEX Interpreter

```
%%%%
% TEST CASE
%
% Test cases are based on the Channel Tunnel Fire report
%%%%

initSpec([action(init),action(stop(t1)),action(decouple(t1)),
          (moving(t1,null))&(neg(norm)) =>o(evacuate(p)),
          modal(stop(t1), moving(t1,null)),
          modal(stop(t1), io(decouple(t1)))]).

initScenario([
              norm,
              p(init),
              modal(init,neg(p(init))),
              moving(t1,null)=>p(decouple(t1)),
              moving(t1,west),
              modal(init,io(stop(t1))),
              neg(p(decouple(t1))),
              moving(t2,null)]).

actions([init,stop(t1),decouple(t1)]).

%%%%
% END OF TEST CASE
%%%%

%-test
%Attempting to solve test models the test case.
```

```
%It gathers the values and passes them to the behaviour
%procedure. The initial scenario is concatenated to the
%specification to produce the full initial scenario.
%This can be thought of as the necessitation rule.
%The relevant P-structure is then assembled using
%the addDeontics procedure.

test:-
    initSpec(A),
    initScenario(B),
    actions(C),
    write('Initial Scenario: '),nl,
    write('Spec: [') ,pp(A), write(']'),nl,
    concatenate(A,B,DB),
    addPermissions(DB,DB1),
    addPrescriptions(DB1,DB2),
    write('Scenario: [') ,pp(DB2), write(']'),nl,
    write('Action Trace: [') ,pp(C), write(']'),nl,
    behaviour(A,DB2,C).


%-behaviour
%This procedure oversees the database manipulation
%from one scenario to the next.
%It takes the action trace ([H|T]), the current
%specification(SP) and the current database (DB).
%Once all the manipulations have been performed for
%the current scenario, the recursive call models movement
%to the next scenario.
behaviour(_,_,[]):-nl,write('end').
behaviour(SP,DB,[H|T]):-
          getEffects(H,DB,DB,DB1),
          checkNorm(DB1),
          sortConflicts(H,DB1,DB1,DB2),
          removeNegs(SP,DB2,DB2,DB3),
          removeOldNorms(DB3,DB4),
          addPermissions(DB4,DB5),
          addPrescriptions(DB5,DB6),
          nl,write('Following:'),write(H),nl,
          nl,write('Spec: [') ,pp(SP),    write(']'),
          nl,write('NewDB: [') ,pp(DB6), write(']'),nl,
          behaviour(SP,DB6,T).

%-getEffects
%This rule determines how action performed to reach the
```

```
%current scenario has affected the scenario.

getEffects(A,B,C,E):-
        getModEffects(A,B,C,D),
        getFOLEffects(A,D,D,E).

%- getModEffects
%This procedure adds to the scenario database all the
%values that are the explicit effects of performing
%the action (A). Where the property already holds,
%it is not added.

getModEffects(_,DB,[],[]).

getModEffects(A,DB,[modal(A,(modal(A,Z)))|T],
        [modal(A,Z),modal(A,modal(A,Z))|T2]):-
    \+member(modal(A,Z),DB),
    getModEffects(A,DB,T,T2).
getModEffects(A,DB,[modal(A,(modal(A,Z)))|T],
        [modal(A,modal(A,Z))|T2]):-
    member(modal(A,Z),DB),
    getModEffects(A,DB,T,T2).

getModEffects(A,DB,[modal(A,Y)|T],[modal(A,Y)|T2]):-
    \+(Y=modal(A,Z)),
    \+member(Y,DB),
    getModEffects(A,[Y|DB],[Y|T],T2).
    %Effects of performing the action.
getModEffects(A,DB,[modal(A,Y)|T],[modal(A,Y)|T2]):-
    \+(Y=modal(A,Z)),
    member(Y,DB),
    getModEffects(A,DB,T,T2).%Effects already present

getModEffects(A,DB,[H|T],[H|T2]):-
    \+H=modal(A,Y),
    getModEffects(A,DB,T,T2).

%- getFOLEffects
%This procedure breaks down the expressions in the
%database to calculate the atomic properties that
%can currently be derived.

getFOLEffects(_,D,[],[]).

getFOLEffects(A,D,[X=>Y|T],[X=>Y|T2]):-
    member2(X,D),
```

```
    member2(Y,D),
    getFOLEffects(A,D,T,T2).
getFOLEffects(A,D,[X=>Y|T],[X=>Y|T2]):-
    member2(X,D),
    \+ member2(Y,D),
    getFOLEffects(A,[Y|D],[Y|T],T2).
getFOLEffects(A,D,[X=>Y|T],[X=>Y|T2]):-
    \+ member2(X,D),
    getFOLEffects(A,D,T,T2).

getFOLEffects(A,D,[X&Y|T],T2):-
    member2(X,D),
    member2(Y,D),
    getFOLEffects(A,D,T,T2).
getFOLEffects(A,D,[X&Y|T],T2):-
    \+member2(X,D),
    \+member2(Y,D),
    getFOLEffects(A,[X,Y|D],[X,Y|T],T2). %
getFOLEffects(A,D,[X&Y|T],T2):-
    member2(X,D),
    \+member2(Y,D),
    getFOLEffects(A,[Y|D],[Y|T],T2).
getFOLEffects(A,D,[X&Y|T],T2):-
    \+member2(X,D),
    member2(Y,D),
    getFOLEffects(A,[X|D],[X|T],T2).

getFOLEffects(A,D,[X|T],[X|T2]):-
    \+(X=(P=>Q)),
    \+(X=(P&Q)),
    getFOLEffects(A,D,T,T2).

%-member2
%This procedure examines the database to
%evaluate if some formula holds.
member2(Goal,DB):-
                member(Goal,DB).
member2(Goal1 & Goal2,DB):-
                member2(Goal1,DB),
                member2(Goal2,DB).
member2(Goal1 or _,DB):-
                member2(Goal1,DB).
member2(_ or Goal2,DB):-
                member2(Goal2,DB).
member2(Goal1=>Goal2):-
                member2(\neg(Goal1) or Goal2,DB).
```

```
member2(neg(Goal),DB):- \+member2(Goal,DB).

%-RemoveNegs
%This procedure removes all the explicit negations
%from the database. Where these contradict scenario
%properties, these properties are also removed.
%Where these contradict specification properties,
%the error is flagged and no change is made.
removeNegs(_,_,[],[]).
removeNegs(SP,DB,[neg(P)|T],T2):-
      member(P,SP),
      write('Unable to contradict Specification Property'),
      nl,
      removeNegs(SP,DB,T,T2).
removeNegs(SP,DB,[P|T],T2):-
      \+(P=neg(Q)),
      member(neg(P),SP),
      write('Unable to contradict Specification Property'),
      nl,
      removeNegs(SP,DB,T,T2).

removeNegs(SP,DB,[P|T],T2):-
      \+(P=neg(Q)),
      member(neg(P),DB),
      \+member(neg(P),SP),
      removeNegs(SP,DB,T,T2).
      %If a previous scenario property has
      %been negated, it is removed.

removeNegs(SP,DB,[neg(P)|T],T2):-
      \+member(P,SP),
      removeNegs(SP,DB,T,T2).

removeNegs(SP,DB,[P|T],[P|T2]):-
      \+member(neg(P),DB),
      \+(P=neg(Q)),
      removeNegs(SP,DB,T,T2).

%-SortConflicts
%This procedure checks for conflicts in the
%database. This procedure checks that new properties
%replace old ones in the DB.

sortConflicts(A,DB,[],[]).
sortConflicts(A,DB,[moving(X,Y)|T],T2):-
        member(modal(A,moving(X,Z)),DB),
```

```
        \+(Z=Y),
        sortConflicts(A,DB,T,T2).%Remove if old  value

sortConflicts(A,DB,[moving(X,Y)|T],[moving(X,Y)|T2]):-
        \+member(modal(A,moving(X,Z)),DB),
        sortConflicts(A,DB,T,T2).
        %Store if no conflicting value.

sortConflicts(A,DB,[moving(X,Y)|T],[moving(X,Y)|T2]):-
        member(modal(A,moving(X,Y)),DB),
        sortConflicts(A,DB,T,T2). %Store if new value.

sortConflicts(A,DB,[H|T],[H|T2]):-
        \+(H=moving(X,Y)),
        sortConflicts(A,DB,T,T2).

%-addPermission
%This procedure calls the appropriate procedure,
%depending on whether there is an immediate obligation
%in the current scenario or not.
addPermissions(DB, NewDB):-
    %If there is an immediate obligation
    member(io(X),DB),
    alterIPs(X,DB,NewDB).
addPermissions(DB,NewDB):-
    %if there is no immediate obligation
    \+member(io(X),DB),
    addIPs(DB,NewDB).

%-alterIPs
%This procedure constructs the P-structure for the
%scenario in which an immediate obligation holds. Also
%implements the negation of the immediate obligation
%in the following scenario.
alterIPs(X,[],[ip(X),modal(X,neg(io(X)))]).
alterIPs(X,[ip(P)|T],T2):-
     alterIPs(X,T,T2).
     %All other immediate permissions are removed.
alterIPs(X,[H|T],[H|T2]):-
     \+(H=ip(P)),
     \+(H=io(Q)),
     alterIPs(X,T,T2).
alterIPs(X,[H|T],T2):-
     (H=io(P)),\+(P=X),write('Error: Two Obligations'),
     alterIPs(X,T,T2).
     % Removes latter if two immediate
```

```
     % obligations in same scenario.
alterIPs(X,[io(X)|T],[io(X)|T2]):-
     alterIPs(X,T,T2).
     % Keep the immediate obligation.

%-addIPs
%This procedure constructs the P-structure for scenarios
%without immediate obligations.
addIPs([],[]).
addIPs([p(X)|T],[ip(X),p(X)|T2]):-
    addIPs(T,T2).
    %Add immediate permissions,
    %if there are weak permissions
addIPs([X|T],[X|T2]):-
    \+(X=p(X)),
    \+(X=ip(Y)),
    addIPs(T,T2).
    %Keep non-permission related statements
addIPs([ip(X)|T],T2):-
   addIPs(T,T2).
   %Remove old immediate permissions.

%-checkNorm
%Simple procedure to return a flag when the system
%is in a non-normative scenario.
checkNorm(DB):-
     \+member(norm,DB),
     write('In non-Normative Scenario'),nl,nl.
checkNorm(DB):-member(norm,DB).

%-removeOldNorms
%This procedure removes old action descriptions that
%describe the accessibility of normative and
%non-normative scenarios from the previous scenario.
removeOldNorms([],[]).
removeOldNorms([H|T],[H|T2]):-
     \+H=modal(_,norm),
     \+H=modal(_,neg(norm)),
      removeOldNorms(T,T2).
removeOldNorms([modal(_,norm)|T],T2):-
      removeOldNorms(T,T2).
removeOldNorms([modal(_,neg(norm))|T],T2):-
      removeOldNorms(T,T2).

%-addPrescriptions
%This procedure adds action descriptions that describe
```

```
%the accessibility of normative and non-normative
%scenarios from the current scenario.
addPrescriptions(DB,DB):- \+member(norm,DB).
addPrescriptions(DB,DB2) :-
      member(norm,DB),
      addMods(DB,DB,DB2).
addMods([action(X)|T],DB,[modal(X,norm)|T2]):-
      member(ip(X),DB),
      addMods(T,DB,T2).
addMods([action(X)|T],DB,[modal(X,neg(norm))|T2]):-
      \+member(ip(X),DB),
      addMods(T,DB,T2).
addMods([H|T],DB,T2):-
      \+H=action(X),
       addMods(T,DB,T2).
addMods([],DB,DB).



%-pp
%This is a rather simple pretty printing procedure that
%ensures no more than two database elements are written
%on the same line.
pp([H,H2,H3|T]):-
              nl,
              write('          '),
              write(H),
              write(', '),
              write(H2),
               write(', '),
               pp([H3|T]).
pp([H,H2]):-
        nl,
        write('          '),
        write(H),
        write(', '),
        write(H2).
pp([H]):-
        nl,
        write('          '),
        write(H).
```

Appendix H

DALEX Example Trace

```
Initial Scenario:
Spec: [
        action(init), action(stop(t1)),
        action(decouple(t1)),
        moving(t1,null)&neg(norm)=>o(evacuate(p)),
        modal(stop(t1),moving(t1,null)),
        modal(stop(t1),io(decouple(t1)))]
Scenario: [
        modal(init,norm), modal(stop(t1),neg(norm)),
        modal(decouple(t1),neg(norm)), action(init),
        action(stop(t1)), action(decouple(t1)),
        moving(t1,null)&neg(norm)=>o(evacuate(p)),
        modal(stop(t1),moving(t1,null)),
        modal(stop(t1),io(decouple(t1))), norm,
        ip(init), p(init),
        modal(init,neg(p(init))),
        moving(t1,null)=>p(decouple(t1)),
        moving(t1,west), modal(init,io(stop(t1))),
        neg(p(decouple(t1))), moving(t2,null)]
Action Trace: [
        init, stop(t1),
        decouple(t1)]

Following:init
Spec: [
        action(init), action(stop(t1)),
        action(decouple(t1)),
        moving(t1,null)&neg(norm)=>o(evacuate(p)),
```

```
        modal(stop(t1),moving(t1,null)),
        modal(stop(t1),io(decouple(t1)))]
NewDB: [
        modal(init,neg(norm)), modal(stop(t1),norm),
        modal(decouple(t1),neg(norm)), action(init),
        action(stop(t1)), action(decouple(t1)),
        moving(t1,null)&neg(norm)=>o(evacuate(p)),
        modal(stop(t1),moving(t1,null)),
        modal(stop(t1),io(decouple(t1))), norm,
        modal(init,neg(p(init))),
        moving(t1,null)=>p(decouple(t1)),
        moving(t1,west), modal(init,io(stop(t1))),
        io(stop(t1)), moving(t2,null),
        ip(stop(t1)), modal(stop(t1),neg(io(stop(t1))))]

Following:stop(t1)

Spec: [
        action(init), action(stop(t1)),
        action(decouple(t1)),
        moving(t1,null)&neg(norm)=>o(evacuate(p)),
        modal(stop(t1),moving(t1,null)),
        modal(stop(t1),io(decouple(t1)))]
NewDB: [
        modal(init,neg(norm)), modal(stop(t1),neg(norm)),
        modal(decouple(t1),norm), action(init),
        action(stop(t1)), action(decouple(t1)),
        moving(t1,null)&neg(norm)=>o(evacuate(p)),
        modal(stop(t1),moving(t1,null)),
        moving(t1,null), modal(stop(t1),io(decouple(t1))),
        io(decouple(t1)), norm,
        modal(init,neg(p(init))),
        moving(t1,null)=>p(decouple(t1)),
        p(decouple(t1)), modal(init,io(stop(t1))),
        moving(t2,null), modal(stop(t1),neg(io(stop(t1)))),
        ip(decouple(t1)),
        modal(decouple(t1),neg(io(decouple(t1))))]

Following:decouple(t1)

Spec: [
action(init), action(stop(t1)),
action(decouple(t1)),
moving(t1,null)&neg(norm)=>o(evacuate(p)),
modal(stop(t1),moving(t1,null)),
modal(stop(t1),io(decouple(t1)))]
```

```
NewDB: (
        modal(init,neg(norm)), modal(stop(t1),neg(norm)),
        modal(decouple(t1),norm), action(init),
        action(stop(t1)), action(decouple(t1)),
        moving(t1,null)&neg(norm)=>o(evacuate(p)),
        modal(stop(t1),moving(t1,null)),
        moving(t1,null), modal(stop(t1),io(decouple(t1))),
        norm, modal(init,neg(p(init))),
        moving(t1,null)=>p(decouple(t1)), ip(decouple(t1)),
        p(decouple(t1)), modal(init,io(stop(t1))),
        moving(t2,null), modal(stop(t1),neg(io(stop(t1)))),
        modal(decouple(t1),neg(io(decouple(t1))))]

end
yes
```