Safety Assurance in Interlocking Design

Matthew John Morley

Doctor of Philosophy
University of Edinburgh
1996

Abstract

This thesis takes a pedagogical stance in demonstrating how results from theoretical computer science may be applied to yield significant insight into the behaviour of the devices computer systems engineering practice seeks to put in place, and that this is immediately attainable with the present state of the art. The focus for this detailed study is provided by the type of solid state signalling systems currently being deployed throughout mainline British railways. Safety and system reliability concerns dominate in this domain. With such motivation, two issues are tackled: the special problem of software quality assurance in these data-driven control systems, and the broader problem of design dependability. In the former case, the analysis is directed towards proving safety properties of the geographic data which encode the control logic for the railway interlocking; the latter examines the fidelity of the communication protocols upon which the distributed control system depends.

The starting point for both avenues of attack is a mathematical model of the interlocking logic that is derived by interpreting the geographic data in process algebra. Thus, the emphasis is on the semantics of the programming language in question, and the kinds of safety properties which can be expressed as invariants of the system's ongoing behaviour. Although the model so derived turns out to be too concrete to be effectual in program verification in general, a careful analysis of the safety proof reveals a simple co-induction argument that leads to a highly efficient proof methodology. From this understanding it is straightforward to mechanise the safety arguments, and a prototype verification system is realised in higher-order logic which uses the proof tactics of the theorem prover to achieve full automation.

The other line of inquiry considers whether the integrity of the overall design that coordinates the activities of many concurrent control elements can be compromised. Therefore, the formal model is developed to specifically answer safety-related concerns about the protocol employed to achieve distributed control in the management of larger railway networks. The exercise reveals that moderately serious design flaws do exist, but the real value of the mathematical model is twofold: it makes explicit one's assumptions about the conditions under which the faults can and cannot be activated, and it provides a framework in which to prove a simple modification to the design recovers complete security at negligible cost to performance.

Acknowledgements

A PhD thesis is seldom completed in isolation, but it is often a lonely activity. My thanks, therefore, to George Cleland and Stuart Anderson in Edinburgh: George, for his cheerful optimism throughout, and Stuart not least for teaching me Iyengar's asanas and the merit of standing on my head. I think this work has benefited from discussions with them, but certainly many of my ideas have been enriched by their experience and that of others in the department.

Outside Edinburgh, I am very grateful to Graham Birtwistle for his encouragement as I struggled to write up this work and without which I should not have completed the job. Thanks, too, to Chris Tofts and Faron Moller for their quiet, moral support (mostly quiet, as we exasperate each other on occasion as friends do.)

Ian Mitchell at British Rail Research contributed much to my knowledge of railway signalling. I will always be a novice, but hopefully he does not feel misrepresented by this text. Last, but by no means least, my thanks go to Axel Poigné for his tacit approval since some of my time was also his.

Declaration

This thesis was composed by myself, and the work it contains is my own except where I have indicated otherwise.

Some of the material in Chapter 5 appeared in the proceedings of the Sixth International Workshop on Higher-order Logic Theorem Proving and its Applications, Vancouver, 1993 [75]; the results reported in Chapter 6 will appear in the *Science of Computer Programming* journal, late in 1996.

Table of Contents

List of Figures v

Notational Conventions vii

1. Introduction 1

1.1 Motivation . . . 1
1.2 A Whistle-stop Tour of Railway Signalling . . . 8
1.3 Solid State Interlocking . . . 11
1.3.1 Overall System Architecture . . . 11
1.3.2 Generic SSI Software . . . 13
1.3.3 Examples of Geographic Data . . . 15
1.4 Inter-SSI Communications . . . 18
1.4.1 Setting Routes over Boundaries . . . 19
1.4.2 Releasing Sub-routes over Boundaries . . . 20
1.4.3 Implementing Remote Route Locking . . . 21
1.5 Formal Approaches to Signalling Safety . . . 22
1.5.1 Related Work . . . 22
1.5.2 Contributions & Thesis Overview . . . 26

2. The Geographic Data Language 29

2.1 Introduction . . . 29
2.2 Static Data and Dynamic Data . . . 30
2.2.1 Geographic Data Identity Files . . . 30
2.2.2 Source Files: Periodic Access . . . 31
2.2.3 Source Files: Random Access . . . 32
2.3 Geographic Data Source File Syntax . . . 34
2.3.1 Examples: Route Locking & Release . . . 35
2.3.2 Concrete Syntax of the Geographic Data Language . . . 36
2.4 Semantics: The Control Interpreter . . . 39
2.4.1 Abstract Syntax of Simple Tests and Commands . . . 39
2.4.2 Points Free to Move Conditions . . . 40

2.4.3 The Map Search 41
2.5 Indirect Semantics of the Map Search 42
2.6 Summary 45

3. Modelling Solid State Interlocking **47**
3.1 Introduction 47
3.2 CCS Model of Solid State Interlocking 50
3.2.1 Modelling Assumptions 50
3.2.2 Model 52
3.2.3 Translating Geographic Data into CCS 54
3.3 Defining Safety Properties Formally 56
3.3.1 Safety Properties of Geographic Data 56
3.3.2 Tags and Probes 58
3.3.3 Geographic Data Invariants 59
3.3.4 Generalising the Translation Schema 61
3.4 The Problem with State Spaces 62
3.4.1 Hiding Assumptions 62
3.4.2 Agent Transformations 63
3.4.3 Model Checking 65
3.5 Proof by Program 67
3.5.1 Generating States of SSI 67
3.5.2 Checking Properties 70
3.6 Summary 71

4. Proving Safety Properties of Geographic Data **74**
4.1 Introduction 74
4.2 Tableau Proofs in Local Model Checking 76
4.2.1 Unfolding Proof Tableaux 76
4.2.2 Partial Tableaux 79
4.3 Invariance & Co-induction 82
4.4 Checking Interlocking Data 85
4.4.1 Sub-route Release Data 85
4.4.2 Route Request Data 87
4.4.3 Unprovable Assertions 89
4.5 From Rigorous to Formal Proofs 91
4.5.1 The Temporal Logic of Actions 91
4.5.2 Unity 93
4.5.3 Floyd-Hoare Logic 95
4.6 Summary 96

5. A Formal Theory of the Geographic Data Language **99**
5.1 Introduction . . . 99
5.2 Geographic Data in Higher-order Logic . . . 102
5.2.1 A Simple Imperative Language . . . 102
5.2.2 Semantics in Higher-order Logic . . . 104
5.2.3 Hoare Logic: Rules and Tactics . . . 107
5.3 A Theory of Geographic Data Invariants . . . 110
5.3.1 Track Circuits – **MX** . . . 111
5.3.2 Points – **PT** . . . 112
5.3.3 Routes – **RT** . . . 114
5.4 Mechanising the Invariance Proof . . . 117
5.4.1 Sub-route Release Data Tactic . . . 118
5.4.2 Route Request Data Tactic . . . 119
5.4.3 Failed Tactics . . . 121
5.5 Decomposing Global Invariance . . . 123
5.5.1 Computational Complexity (Revisited) . . . 123
5.5.2 Heuristics for Decomposition in the Proof . . . 125
5.5.3 Static & Dynamic Decomposition . . . 126
5.6 Summary . . . 128

6. Distributed Control in Complex Interlockings **131**
6.1 Introduction . . . 131
6.2 The Remote Route Request Protocol . . . 135
6.2.1 Preliminaries: Elapsed Timers and Telegrams . . . 135
6.2.2 Geographic Data . . . 136
6.2.3 Safety Considerations . . . 139
6.3 Modelling Remote Route Locking . . . 142
6.3.1 Timing Issues . . . 142
6.3.2 A Formal CCS Model . . . 142
6.3.3 Matching up the Interfaces between East & West . . . 145
6.3.4 Axiomatising Remote Route Requests . . . 146
6.4 Safety Properties of the Model . . . 148
6.4.1 First Refinement: Eliminating Arbitrary Delays . . . 149
6.4.2 Second Refinement: Adding Priorities . . . 151
6.4.3 Lossy Communications and Duplicating Telegrams . . . 152
6.5 Summary . . . 155

7. Safety in Interlocking Design **157**
7.1 Implementing Remote Route Locking Safely 157
7.2 Leamington Spa 162
7.2.1 Strengthening the Invariant 162
7.2.2 Swinging Overlaps 163
7.3 Conclusions 167
7.3.1 Theorem Proving 167
7.3.2 Semantics 169
7.3.3 Model Checking 171
7.3.4 Railway Signalling 173

Bibliography **175**

A. Glossaries **182**
A.1 Glossary of Railway Signalling Terms 182
A.2 Glossary of SSI Terminology 187
A.3 Glossary of Geographic Data Terminology 191

B. Theory **195**
B.1 Calculus of Communicating Systems 195
B.2 Modal μ-calculus 200
B.3 HOL Proofs 207

C. Examples of Geographic Data **210**

List of Figures

1.1 Part of a signal control panel 10
1.2 Schematic overview of the main features of SSI 13
1.3 Scheme plan with route and sub-route annotations 16
1.4 EAST-WEST—Setting routes divided by Interlocking boundaries . . . 19

2.1 Signalling scheme plan for WEST 34
2.2 Geographic Data: conditional language constructs 36
2.3 Semantics of the conditional language 45

3.1 Trailing points may derail trains 48
3.2 Simple grammar for a subset of the Geographic Data Language 51
#1 A CCS model of Solid State Interlocking 53
3.3 Translating Geographic Data into CCS 54
3.4 Panel route request `*Q02` translated into CCS 55
#2 An observable model of SSI 59
3.5 Geographic Data invariant for WEST 60
3.6 Transition rules for *PRR* and *FOP* data 68
3.7 Results of Proof by Program 71

4.1 A Partial tableau 80
4.2 Routes R_2 and R_{51} from the scheme plan for WEST 90

5.1 Denotational Semantics of Geographic Data Language Commands . . 104
5.2 HOL Derived Rules of Floyd-Hoare Logic 109
5.3 Routes that diverge after a common segment 115
5.4 *Routes*: Massaging the **RT** invariant 117
5.5 Experiments using HOL on some simple Geographic Data 123
5.6 A distinction between network structure and route structure 127

6.1 EAST-WEST—Setting routes across SSI boundaries 132
6.2 Train derailment due to partial route setting 134
6.3 Normal sequence of events in making remote route requests 140

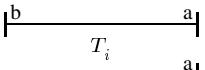
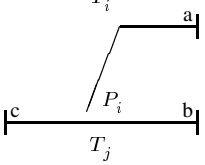
6.4 Abnormal sequence of events in making remote route requests 141
#3 Simple model of SSI communications over Internal Data Link 146
6.5 Generic rules for remote route locking and release 147
#4 Refining Model #3 so as to discard tardy IDL inputs after Δ cycles . . 149
6.6 Illustrating how unsafe states arise in Model #4 when $\Delta = 2$ 151
#5 Introducing lossy link behaviour to Model #4 152
#6 Refining Model #5 to filter duplicate IDL inputs 154

7.1 Modified rules for remote route locking and release 159
7.2 Final version of the Internal Data Link model in CCS 161
7.3 Sample *PRR* data from Leamington Spa 163
7.4 Overlaps 164

B.1 Transition rules for pure CCS 196

C.1 Sample interlocking: WEST 210
C.2 The EASTWEST interlocking 213
C.3 The FOREST LOOP interlocking 215
C.4 The THORNTON JN. interlocking 216
C.5 An artistic impression of Leamington Spa 217

Notational Conventions

	Plain (bi-directional) track section
	Points track section (points shown normal)
	Three aspect main signal (red/green/yellow)
	Two aspect main signal (red/green)
T_i	Track circuit identifier (for track section T_i)
S_i	Signal identifier
P_i	Points identifier
T_i^{ab}	Sub-route over T_i in the direction $\vec{ab}$
R_{mn}	Route from signal m to signal n
R_m	Route from signal m into another interlocking
Z_i^{ab}	Dummy sub-route over T_i in the direction $\vec{ab}$
O_i^{ab}	Sub-overlap over T_i in the direction $\vec{ab}$
$\mathcal{P}$	Set of points identifiers (or its cardinality)
$\mathcal{R}$	Set of route identifiers (or its cardinality)
$\mathcal{S}$	Set of signal identifiers (or its cardinality)
$\mathcal{T}$	Set of track circuits (or its cardinality)
$\mathcal{U}$	Set of sub-routes (or its cardinality)
$\mathcal{Q}$	Set of panel (route) requests
FOP	Flag operations data file
PRR	Panel route request data file
PFM	Points “free to move” data file
IPT	Input telegram data file
OPT	Output telegram data file
MAP	Map search data file
`*L`, `*Q`, `*PN`, `*PR`	Labels in Geographic Data source code
`@L`, `{L`, `}L`, `^L`	Label references

Chapter 1

Introduction

This thesis presents a study into the application of theoretical computer science to problems arising in the railway signalling industry. Although the focus is on the type of computer controlled signalling systems that are currently replacing electromechanical technology throughout mainline British railways, the analysis techniques used and illustrated here are of a general nature and may be applied in a similar fashion to a range of data-driven control systems. The technical material in this thesis is presented in a style which, it is hoped, is sufficiently transparent to be intelligible to practising engineers seeking to emulate the study. This introductory chapter covers much of the background needed to establish an intuitive framework which later chapters will build upon more formally.

1.1 Motivation

While results from this work have already enjoyed both direct and indirect influence in the given industrial domain, this thesis is not *about* railway signalling any more than it is *about* computer science itself. Indeed, this work falls somewhere between the needs of engineering practice on the one hand, and the advancing scientific basis of computing on the other. That it does so is not really an accident: it is precisely because of the gulf that exists between the communities of industrial developers of computer based systems whose work is strongly governed by market imperatives, and academic computer scientists who have hitherto been motivated more by the mathematical elegance and precision of their theories. Our endeavour is, in a small way, to shed some light on what lies in between these positions with a particular regard to the application of notions from theoretical computer science to relevant problems faced by industry.

Applied theoretical computer science has, for better or worse, become synonymous with the term 'formal methods'. Despite several decades of research in the area, formal methods have yet to be wholeheartedly incorporated into the development of computer systems on any scale of design. In the large scale applications that include the control

of industrial plant, power generation, aviation and mass transportation, this may be because the move towards computer dominated solutions to the engineering problems is a relatively recent development for communities and licensing authorities that have strong, conservative safety cultures. On the smaller scale, in consumer electronics say, the financial risks seem too great when formal mathematical techniques towards software development and system design are difficult to apply in general, currently impossible to use with only naïve understanding of the theory and supporting tools, and poorly promoted by real, or even realistic, case-studies from which to learn.

The emergence of two relevant industrial standards is thus particularly interesting: *MOD 00-55: The Procurement of Safety Critical Software in Defence Equipment* [69], and *IEC 1131-3: Software for Computers in the Application of Industrial Safety-Related Systems* [46]. One of the requirements appearing in the former (Interim) standard is the mandatory use of formal methods in projects intended to supply equipment to the UK Ministry of Defence; the latter standard defines several languages (*e.g.*, Function Block Diagrams, or Structured Text) that provide software for Programmable Logic Controllers (PLC), mainly used in the process control industry. Neither standard is solely for use within the given sector. However, it is fair to say that the *assemblage* of PLC languages defined in IEC 1131-3 seems insufficiently well defined mathematically for such software to be readily acceptable according to the coding requirements of MOD 00-55. This is a shame since the relative simplicity of the PLC languages (*e.g.*, when compared to Ada) ostensibly offer excellent opportunities for the kind of formal design and development anticipated in MOD 00-55.

Interestingly, Halang and Krämer [39] (unintentionally) illustrate one of the key technical difficulties involved here. Their work addresses the reluctance of licensing authorities to certify software embedded in process control with a proposal for a formal software development methodology for PLC programs. The underlying theory is OBJ through which the authors formalise requirements and functional aspects of the design specification assembled as a Function Block Diagram. Properties of the specification can be verified largely automatically with the mechanical support for OBJ, and design validation is further supported through symbolic execution. A Structured Text program (Structured Text is a small, procedural tasking language) is then developed and annotated in the sense of Floyd-Hoare logic with assertions drawn from the requirements and specification documents—the verification conditions derived from the structured text can also be discharged using OBJ. Practical considerations aside, the main problem with this development methodology is simply that the relationship between Function Block Diagrams and Structured Text (and indeed the other PLC languages) is entirely *informal*. This reflects the relationship between these languages in the defining IEC standard.

Halang and Krämer seriously tackle the crisis in embedded software but there may yet be some doubt, depending of course on how one interprets the standard, whether their efforts satisfy the strictures of MOD 00-55. (Similar concerns have also been voiced about the RAISE wide spectrum language and programming methodology [23].) While there is no intended cross-reference between the two cited standards, the point is that neither really grasps the strength of applied theoretical computer science. MOD 00-55 is very rigid in its definition of 'formal methods' and probably overestimates the benefit currently to be accrued from, if not the difficulty attending to, the use of mathematical formalism in *developing* software all the way down from high-level requirements to detailed code. Although MOD 00-55 addresses software issues only, neither it nor MOD 00-56 [70] with which it is explicitly related, recognise a potential rôle for formal methods in supporting system design as a whole. The IEC standard, however, underestimates the insight and assurance to be obtained by theoretical analysis of programming languages, by clarifying their mathematical definitions and, in this case, by elaborating the semantics of their interactions. This is a particular concern for languages intended for use in the burgeoning area of safety critical computer systems.

Mandating the use of formal methods is certainly one way of getting designers to use them, but the paucity of guidance of the *how-to* variety is evidently a major stumbling block: the guidance [68] offered on the selection of formal techniques to use to meet the requirements of MOD 00-55 suggests that case-studies published in the open literature should already demonstrate their successful industrial application. In 1991 such evidence was thin on the ground even for 'mainstream' formal methods like *Z* and VDM; several years later the situation is not much improved, but the publication of the NIST report by Craigen, Gerhart, and Ralston [22, 23], and the FAA report by Rushby [83], indicate growing awareness in several key industrial sectors. The former was commissioned by the (US) National Institute of Science and Technology (and other bodies) to inform deliberations within industry and government on the potential impact of formal methods on standards and regulations; the latter report was commissioned by the (US) Federal Aviation Administration who face the increasingly challenging task of certifying to very high levels of dependability the computer systems on board commercial aircraft (in particular).

The NIST report summarises twelve industrial applications of formal methods used with varying degrees of mathematical rigour on projects of substantial commercial importance. The timescales involved ranged from about nine months (Hewlett-Packard, Medical Instruments), to about nine years (GEC Alsthom, Railway Signalling). The projects studied focused mainly on software specification. The GEC case-study was thought particularly successful: 15,000 lines of formally specified and verified code

were produced for a signalling system that increased the capacity of one Paris Metro line by 12,000 passengers per hour (25%), so saving the operators the enormous cost of constructing a new line to meet the capacity. The success of the project is perhaps better judged by the fact that GEC Alsthom are presently using the same tools and software development techniques on similar railway contracts. The Hewlett-Packard case-study is singled out here for another reason: this produced 4,500 lines of 'zero defect' code (according to the certifier, another branch of the organisation) through machine supported formal specification, but without proof because the software was not safety critical. The project was also intended to achieve technology transfer by promoting the specification language used, but was in this respect an abject failure.

Implicated in the additional costs that arose in the least successful case-study summarised in the NIST report (Ontario Hydro, Tripwire Computer) is the lack of tools support for the formal method used (the fidelity of the formalism is not otherwise in question). Given such a small sample it would be premature to suppose that there is in general a relationship between the level of tool support and the costs incurred (or savings made) in applying formal methods. What several examples in the NIST report do indicate is that the methodology into which more rigorous mathematical techniques are introduced is at least as important to the project's success as the strength of tools applied. However, perhaps the strongest suggestion Craigen *et al.* put forward *is* that the standard of tools supporting particularly the deeper applications (where, for example, proofs demonstrating the conformance of code to specifications are required to satisfy licensing authorities) urgently needs to be improved. This may be so, but begs the question whether there are tangible benefits presently to be obtained through formal verification at the level of program code: the report does not examine the issue, but the evidence presented hardly convinces one that there are—indeed, it rather illustrates that the rôle of formal methods (or *formal proof*) in the design of complex systems is as yet poorly understood. Unfortunately the analysis in the NIST report is quite shallow, and it therefore offers little guidance in these matters.

Rushby [83] on the other hand, offers considerable practical guidance on the uptake of formal methods in a thorough, and far ranging, survey of current techniques in support of system (and software) quality control, and assurance. This report is intended to inform licensing bodies, in particular the FAA, about the strengths and fallibilities of mathematical formalism in system design, *as well as* to inform those seeking to license critical software-based systems about where in the development lifecycle to apply formal methods to best effect. The analysis begins with a classification of formal methods that is given in terms of the degree of rigour attending the mathematical argu-

ments used in support of system development. In a simplified form:

Level 0 No applied mathematics at all, but perhaps appeal to tabular or diagrammatic notations, pseudocode, and equations defining transfer functions, *etc.*.

Level 1 The use of concepts and notations from discrete mathematics, with proofs conducted in the traditional, informal style of mathematical discourse.

Level 2 The use of formalised specification languages with mechanised support for syntax analysis, pretty-printing, and simple type checking.

Level 3 The use of fully formal specification languages with comprehensive support environments including mechanised theorem proving and proof checking.

Proofs at levels 1 and 2 are conducted in the manner of the rigorous arguments preferred by mathematicians, although specification formalisms at level 2 may provide deduction rules that could in principle lead to formalising such arguments; the transition to level 3 is therefore marked by the provision of theorem provers and the 'fully formal' specification languages alluded to which are firmly rooted in mathematical logic (making mechanical support a practical necessity), and which have demonstrably sound axiomatisations.

This classification may not be universally applicable, but it serves Rushby's purpose which is to examine the likely rôle of formal methods in the development lifecycle and in the certification of critical systems. His conclusions address the aviation industry specifically, but are quite unequivocal in asserting that *their* current best practise, based as it is on design reviews, inspections and structured walkthroughs, and buttressed by various approaches to testing to support verification and validation, appear to be adequate for the task of producing certifiable software from clearly stated requirements and unambiguous specifications. Formal methods should first be applied in the earlier stages of the lifecycle, with whatever appropriate degrees of rigour, to produce precise statements of requirements and assumptions, and thoroughly debugged design specifications. Neither the evidence nor the analysis in the FAA report prioritises the application of formal methods to the problem of producing code from specifications.

Rushby goes on to argue that the most rigorous (*i.e.*, level 3) applications of formal methods should be brought to bear precisely where traditional approaches are apparently least adequate: in designing those aspects of digital (avionic) systems that deal with the management of hardware redundancy, algorithms to achieve fault tolerance, and the synchronisation of independent control channels (in particular). The anecdotal evidence in the NIST report also indicates that the lower level applications of formal methods (*e.g.*, RAISE and Z which are identified explicitly) are particularly weak in

the specification and analysis of the coordination between concurrent, synchronous, but often asynchronous and distributed, activities. To apply formal methods with success at any level to such challenging problems demands intense effort and deep abstractions in order to gain intellectual control over the task at hand. But then formal methods can, and indeed as we shall see in later chapters *do*, lead to discoveries and insights into the nature of complex control systems that are quite unapproachable by other means.

Some care has to be exercised in transferring Rushby's analysis to other industrial domains however, since the non-existence of a safe state for airborne systems naturally introduces a heightened appreciation of the need for rigour and robust design throughout the development cycle which may be less marked in other industrial domains. Yet many terrestrial control systems are acutely safety critical and have similar architectural needs that involve replicated hardware for high availability, independent data and control channels and voting mechanisms to mask random environmental perturbations, and so on. Moreover, there is an important class of control systems that apparently challenge the notion that formal verification 'close to the code' does not deserve a high priority: these are *data-driven* control systems. The architecture of such systems typically consists of a generic hardware platform that executes a generic read/write loop (or polling cycle) which is parameterised by application specific control laws manifest in static data that are interpreted to yield appropriate responses to polled inputs. Such data (though one might as well call them software) may be highly safety critical in that they govern the behaviour of the control system as a whole; they therefore demand the most rigorous techniques of analysis (and design).

Large scale examples of data-driven control systems that have been developed in recent years can be found in the railway signalling industry. A specific example is British Rail's *Solid State Interlocking* which is described in more detail in later sections of this chapter. Clearly, one of the main attractions of developing data-driven controllers for highly complex systems such as this is that to a large extent they effect a clean separation of concerns. On the one hand the computer systems engineering concerns, such as those identified by Rushby, are focused in the design of the generic hardware platform and control software. On the other hand the application specific concerns that can be addressed by domain experts without the need for particular computing skills are focused in the preparation of the data. In the world of railway signalling the application data are instantiated for each network installation—that is, roughly speaking, for each station—and express logical relationships between the various controlled elements in the network as well as dynamic relationships between trains and signals, and the sequencing of signal aspects. Such data vary in the details according to the geographic layout of the network concerned; then, the generic control software applied to these *geographic data* yields the desired control function.

Unfortunately a complete separation of concerns according to this division between data and control is rather more ideal than it is commonplace. Complications arise in the realm of railway signalling from the practical necessity of subdividing the railway into separate control authorities. One has therefore to design the interfaces, or *protocols*, between physically separated controllers, and this inevitably blurs the division between what one considers to be application data, and that which is thought of as generic control software. *Some* layer of the protocol will very likely be expressed in the application data since it is required to set up communication to enable one control system to perform specific (within the geography of the network concerned) signalling functions on behalf of another. But if one wishes to avoid programming timers, queues, and watchdogs (the stock in trade of protocol design) at the application layer then one is left with little choice but to encode specific domain knowledge about the nature of the data transferred, or the functions requested/acknowledged, in the underlying architecture. Thus, in order to conceal the interface at the application layer (this might be construed as a good thing to do) one has to complicate the generic software with non-generic code.

While this example illustrates something of a paradox in the philosophy of (distributed) data-driven control, in practice some experimentation leads to a workable compromise. But this thesis will demonstrate, echoing Rushby's appeal for the utmost rigour in precisely this area of system design, that enormous care has to be exercised in building such interfaces. For when in Chapter 6 our formal analysis is focused on one of the protocols by which Solid State Interlocking achieves distributed control of the railway, we shall indeed find subtle (and not-so-subtle) flaws in the overall design.

A second advantage of data-driven controllers is the introduction of application specific languages in which to express the control functions. A well-known example is Ladder Logic (or Ladder Diagrams, now standardised in IEC 1131-3) which evolved in the electrical engineering community as a specification notation for relay circuits (with delayed feedback). Ladder Logic has found use in interlocking design too, but with solid state logic gradually replacing the more costly relay logic, more appropriate notations have begun to emerge. An example is British Rail's *Geographic Data Language* which is studied in this thesis. Both these examples are *expressively weak*: application specific languages are not typically called upon to express more complex programs than sequences of commands like

`IF` ⟨conditions⟩ `THEN` ⟨actions⟩

for simple atomic actions like assigning a variable or setting a register, and conditions that are expressed in terms of internal state variables. From the point of view of software assurance such languages are interesting in several ways. Firstly, because

they support design abstraction in a notation that is closely integrated with the application domain. Secondly, such unsophisticated languages admit precise mathematical definitions from which compilers and interpreters can be rigorously derived. Finally, efficient interpretations in formal logic are possible to realise *automatic* verification tools for checking behavioural properties of their programs.

With respect to Solid State Interlocking in particular, one of the problems with data preparation is that the activity is very much like programming, even to the point that the specifications are incomplete. For signalling, specifications are given by *control tables* which, loosely, indicate all of the conditions that have to be satisfied before a signal can be switched from red to green to admit a train into the track section beyond. These tables have a well defined syntax, and a clear meaning for signalling engineers, but remain exceedingly difficult to 'get right'—so difficult, in fact, that some railway authorities have abandoned control tables as specification documents. Nonetheless these are used along with other documents by British Rail to guide the production of their geographic data. In the absence of any means to demonstrate completeness (in the informal sense, but also the formal) of these specifications there is inevitably a need to *verify* that the derived code does enjoy certain fundamental safety properties—such as logically prohibiting the possibility for two trains to simultaneously enter the same section of the railway.

Traditional methods of verification, *i.e.*, those which constitute current practice, are based on inspections of control tables and the derived geographic data, on simple decompilers and syntax comparators, and massive testing both in the design office and *in situ*. The enormous combinatorial complexity inherent to railway interlockings means that exhaustive simulation is simply impossible. Yet the syntactic nature of the data also make visual inspection an extremely arduous task—so that the discovery by this means of deep errors (problems of specification), or even minor 'typographic' errors (problems of coding), can be haphazard at best. Logical flaws in geographic data *do* emerge through testing the designs, but there is a clearly recognised need (throughout the industry, in fact) to reduce the costs of extensive testing and to boost productivity in interlocking design. To achieve these ends, but particularly to introduce the rigour needed to radically improve quality assurance, calls for a measured introduction of formal methods into the design process. These are some of the reasons why we need theorem proving for geographic data.

1.2 A Whistle-stop Tour of Railway Signalling

Railway signalling engineers face a difficult distributed control problem. Train drivers can know little of the overall topology of the network through which they pass, or of the

whereabouts of other trains in the network and their requirements. Safety is therefore invested in the control system, or ***interlocking*** (the glossary clarifies the meaning of unfamiliar signalling terms emphasised thus), and drivers are required only to obey signals and speed limits. The task of the train dispatcher (signalman, or signal operator) is to adjust the setting of switches and signals to permit or inhibit traffic flow, but the interlocking has to be designed to protect the operator from inadvertently sending trains along conflicting routes.

The network can be operated with more security and efficiency if the operators have a broad overview of the railway and the distribution of trains. Since the introduction of mechanical interlockings in the late 1800's, and as the technology has gradually improved, the tendency has therefore been for control to become progressively centralised with fewer signal control centres individually responsible for larger portions of the network. In the last decade ***Solid State Interlocking*** has introduced computer controlled signalling, but the task of designing a safe interlocking remains essentially unchanged.

Solid State Interlocking is a data-driven signal control system designed for use throughout the British railway system. SSI is a replacement for electromechanical interlockings—which are based on highly reliable relay technology—and has been designed with a view to modularity, improved flexibility in serving the needs of a diversity of rail traffic, and greater economy. The hugely complex relay circuitry found in many modern signalling installations is expensive to install, difficult to modify, and requires extensive housing—but the same functionality can be achieved with a relatively small number of interconnected solid state elements as long as they are individually sufficiently reliable. SSI has been designed to be compatible with current signalling practice and principles of interlocking design, and to maintain the operator's perception of the behaviour and appearance of the control system.

At the signal control centre a ***control panel*** displays the current distribution of trains in the network, the current status of signals, and sometimes that of ***point switches*** (points) and other signalling equipment. The railway layout is depicted schematically on the panel by a graphic similar to Figure 1.1. There are seven (three aspect) main signals shown here, and three sets of points. It is British Rail's practice to associate ***routes*** only with main signals. The operator can select a route by pressing the button at the entrance signal (say, S_7), then pressing the button at the exit signal—the consecutive main signal, being the entrance signal for the next route (S_5). This sequence of events is interpreted as a ***panel route request***, and is forwarded to the controlling computer for evaluation. Other panel requests arise from the ***points keys*** which are used to manually call (and hold) the points to the specified position, or from button pull events (to cancel a route by pulling the entrance signal button).

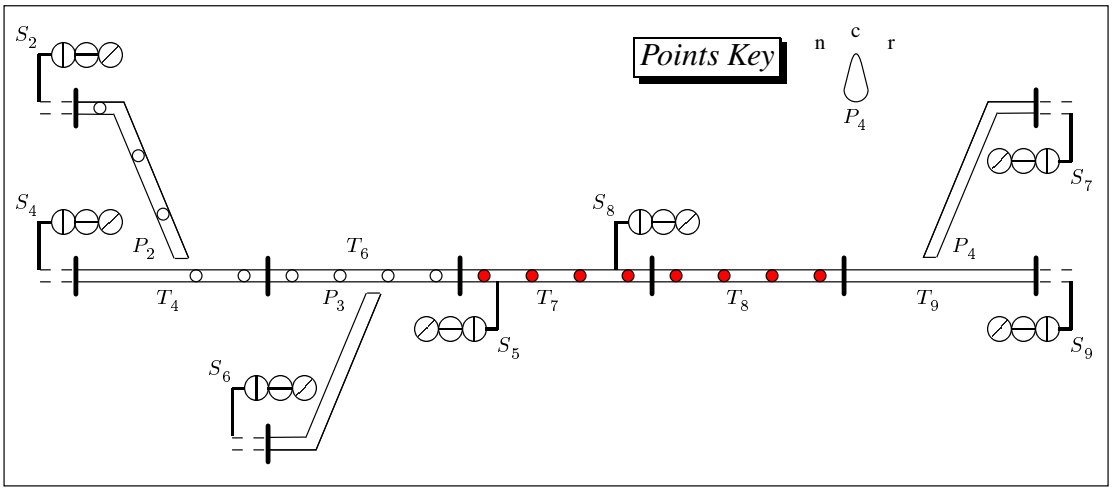


Figure 1.1: Signals (S_i) on the control panel appear on the left to the direction of travel, each signal has a lamp indicator, and each main signal has a button. Switches (points, P_i) show the *normal* position, and there is usually a points key on the panel so one can throw the points 'manually'. Lamps illuminate those track sections (T_i) over which routes are locked (white), and those in which there are trains (red).

When the controlling computer receives a panel route request it evaluates the availability conditions specified for the route. These conditions are given in a database by ***Geographic Data*** which the control program evaluates in its on-going dialogue with the network. If the availability conditions are met the system responds by highlighting the ***track sections*** along the selected route on the display (otherwise the request is simply discarded). At this point the route is said to be *locked*: no conflicting route should be locked concurrently, and a property of the interlocking we should certainly verify is that no conflicting route *can* be locked concurrently.

Once a route is locked the interlocking will automatically *set* the route. Firstly, this involves calling the points along the route into correct alignment. Secondly, the route must be *proved*—this includes checking that points are correctly aligned, that the filaments in the signal lamps are drawing current, and that signals controlling conflicting routes are on (*i.e.*, red). Finally, the entrance signal can be switched off when the route is clear of other traffic—a driver approaching the signal will see it change from red to some less restrictive aspect (green, yellow, *etc.*), and an indicator on the control panel will be illuminated to notify the operators.

The operation of Solid State Interlocking is organised around the concept of a polling cycle. During this period the controlling computer will exchange messages with each piece of signalling equipment to which it is attached. An outgoing command telegram will drive the track-side equipment to the desired state, and an incoming data telegram will report the current state of the device. Signalling equipment is interfaced with the SSI communications system through ***track-side functional modules***. A points module will report whether the switch is *detected normal* or *detected reverse* depending

on which, if either, of the electrical contacts in the switch is closed. A signal module will report the status of the ***lamp proving circuit*** in the signal: if no current is flowing through the lamp filaments the lamp proving input in the data telegram will warn the signal operators about the faulty signal.

Other than conveying status information about points and signals, track-side functional modules report the current positions of trains. These are inferred from ***track circuit*** inputs to the modules. Track circuits are identified with track sections which are electrically insulated from one another. If the low voltage applied across the rails can be detected, this indicates there is no train in the section; a train entering the section will short the circuit causing the voltage to drop and the track section will be recorded as *occupied* at the control centre. Track circuits are simple, fail-safe devices, and one of the primary safety features of the railway.

All actions performed by Solid State Interlocking—whether in response to periodic inputs from the track-side equipment, aperiodic panel requests, or in preparing outgoing command telegrams—are governed by rules given in the Geographic Data that configure each Interlocking differently. Some examples of route locking and release data are explained in Section 1.3.3 below. The ***Geographic Data Language*** (GDL) is introduced in more depth in Chapter 2. In the following section an outline is given of the architecture of the system, and the organisation of the software. These details are needed for a proper appreciation of the models developed in succeeding chapters.

1.3 Solid State Interlocking

Cribbens [24] describes the overall organisation and operation of SSI, and discusses many of the philosophical and technical problems that have had to be overcome in its development. Here we only recall the salient details in order to give a broad overview of the architecture and the manner in which the system maintains safety. The Glossary in Appendix A.2 accompanies this section.

1.3.1 Overall System Architecture

SSI is a multicomputer system with two panel processors, a diagnostic processor, and three central interlocking processors which operate in repairable triple modular redundancy. Higher-order control devices such as route planning and automatic route setting computers are not part of SSI, but they can be interfaced with the system.

The ***central interlocking processors*** are responsible for executing all signalling commands and producing correct system outputs, and operate in TMR to ensure high availability and single fault tolerance in the presence of occasional hardware faults. These are the safety critical elements of SSI. A TMR system has been implemented

for hardware reliability: each subsystem is identical, and runs identical software. All outputs are voted upon, redundantly in each interlocking processor, and the system is designed so that a module will be disconnected in the event of a majority vote against it—SSI will continue to operate as long as the outputs of the remaining modules are in agreement. A replacement module is updated by the two functioning modules before being allowed online. (In the sequel we usually refer to the central interlocking processors collectively as *the SSI*, or *the Interlocking*.)

The ***panel processors*** are responsible for tasks which are not safety critical such as interfacing with the signal control panel, the display, and other systems such as automatic route setting computers. These processors are run in duplex 'hot standby' for reasons of availability. The diagnostic processor is accessible from a maintenance terminal (the ***technician's console***) through which the system's performance and fault status can be monitored, and whereby temporary restrictions on the Interlocking's behaviour can be introduced. In the latter case this is a provision for temporarily barring routes, locking points, or imposing other restrictions that are not directly under the control of the signal operators (for example, at times when there is a need for track maintenance).

A central feature of SSI is that the controlling computer is directly connected to track-side equipment by means of a duplex ***data highway*** carrying discrete signalling information (*cf.* Figure 1.2). Track-side functional modules (TFMs) interface with signals and points to provide power switching under microprocessor control. Here, duplication of the hardware has been designed to ensure safe response to failures, but not fault masking: the TFM will set its outputs to the most restrictive state (*e.g.*, signals at red) whenever a fault is detected or the duplicated control paths are found to diverge. One points module may be connected to two to four point switches, and can report up to four track circuit inputs. A signal module is usually connected to one signal and several nearby track circuits, but is flexible enough for any other desired function.

The operation of Solid State Interlocking is organised around the concept of a ***major cycle***. During this period the central interlocking will address each of the track-side functional modules, and expect a reply from each in turn. A maximum of 63 TFMs can be connected to one SSI, and the major cycle is consequently divided into 64 ***minor cycles***. In the zeroth cycle data are exchanged with the diagnostic processor. In each minor cycle the central interlocking will decode one incoming message (or ***data telegram***) from the data highway, and process one outgoing ***command telegram***.

The cable conveying messages to and from the central interlocking is a screened twisted pair carrying relatively high signal levels. Cribbens discusses in detail the performance requirements for this vital component of the system: the minimum refresh rate for the TFMs, the necessity of real-time encoding and decoding of transmitted

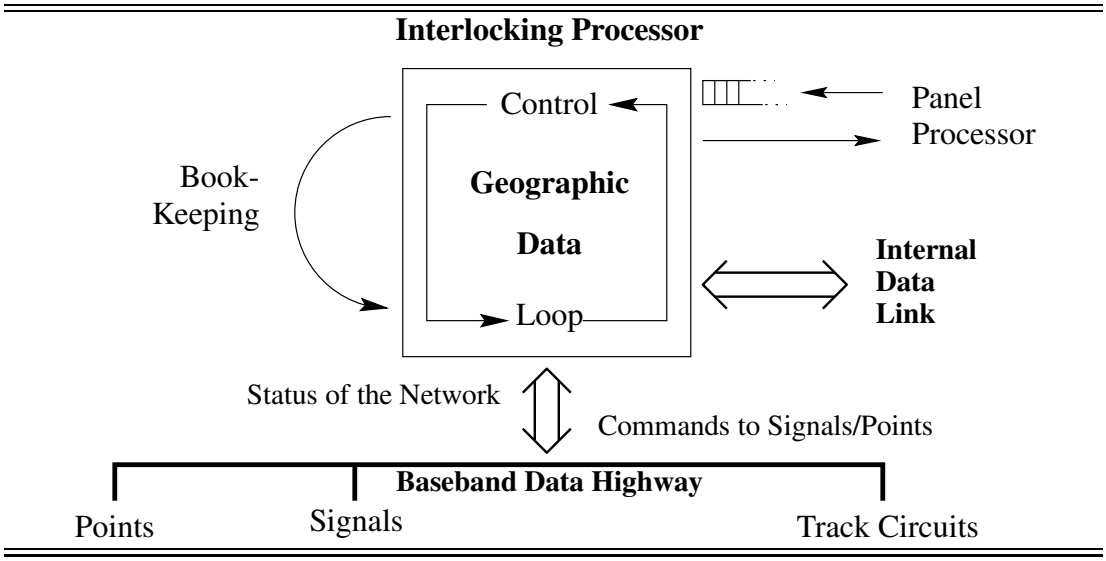


Figure 1.2: Schematic overview of the main features of SSI

data, the geographic extent of the interlocking area and the need for an acceptable range without the need for repeaters (circa 15 km), are all factors that contribute to the design. A data rate of 20k bits per second has been adopted, and a cyclic polling strategy implemented to ensure early detection of communications breakdown at either end of the link. The data path is duplicated and TFMs and central interlocking are designed to tolerate single faults on the line—detected through missed or corrupted messages. In each addressing cycle 25 bits of message data are padded with five parity bits to form a truncated (31,26) Hamming code which is transmitted in Manchester encoded biphase form. TFMs are configured to reply immediately upon receipt of a message from the central interlocking. Cribbens argues convincingly that the SSI transmission system is highly secure.

1.3.2 Generic SSI Software

SSI has been designed to be data-driven with a generic program operating on rules held in a 'geographic' database. These data configure each SSI installation differently, and define the specific interlocking functions (although the more primitive functions are directly supported by the software). The relationship between generic program and the data is one in which the former acts as an *interpreter* for the latter—for this reason we usually refer to the generic software as the ***control interpreter*** in the sequel. The Motorola 6800 microprocessors used in SSI have a 16-bit address space: 60–80k bytes are EPROM which hold the generic program (about 20k bytes), and the Geographic Data; 2k bytes are RAM, and the rest is used for input and output devices. The modest RAM is used, mainly, to hold the system's record of the state of the railway—generally

referred to as the *image of the railway*, or the *internal state* in the sequel.

All SSI software is organised on a cyclic basis with the major cycle determining the rate at which track-side equipment receive fresh commands, and the rate at which the image of the railway is updated. During one minor cycle the generic program: performs all redundancy management, self-test and error recovery procedures; updates system (software) timers and exchanges data with external devices such as panel processors; decodes one incoming data telegram and processes an associated block of Geographic Data; and processes the data associated with one outgoing command telegram. The latter phase is the most computational intensive part of the standard minor cycle because it is through these data that the Interlocking calculates the correct signal aspects.

The SSI minor cycle has a minimum duration of 9.5 ms, and a minimum major cycle time of 608 ms. However, SSI can operate reliably with a major cycle of up to 1,000 ms, with an individual minor cycle extensible to 30 ms. This flexibility is needed for handling panel requests. If the required minor cycle processes mentioned above can be completed in under the minimum minor cycle time, the control interpreter will process one of any pending panel requests (which are stored in a ring buffer). The data associated with a panel request must not require more than a further 20 ms of processing time—the data are structured such that accurate timing predictions can be made at compile time. If the minor cycle is too long the track-side functional modules will interpret the gaps between messages as data link faults, and will drive the equipment to the safe state in error.

The initialisation software compares the internal state of each of the three interlocking processors to determine the required start up procedure. When power is first applied a ***'mode 1' startup*** is necessary: this sets the internal state to a (designated) safe configuration, forces all output telegrams to drive the track-side equipment to the safe state and disables processing of panel requests; after a suitable delay so that TFM inputs can bring the internal state up to date, the Interlocking can be enabled under supervision from the technician's console. After a short power failure much of the contents of RAM will have been preserved and a 'mode 2' or 'mode 3' start up is appropriate. A 'mode 2' start up resets the internal state to the safe configuration but preserves any restrictions that had been applied through the technician's console—the system is disabled for a period long enough for all trains to come to a halt, and allowed to restart normal operation automatically. A 'mode 3' start up involves a similar reset but the status of routes is also preserved, and the system restarts immediately.

Validation of the generic SSI program has been described by Short [85] who points out the need for extensive testing to validate the final hardware and software combination because the software performs safety checks (redundancy management, *etc.*)

on the hardware. Short also notes the difficulty (*i.e.*, the intractability) of performing correctness proofs at the level of the semantics of 6800 assembler—yet when one considers that there are of the order of twenty megabytes of control and monitoring software on board the A340 [5] airliners, for example, the 20 thousand bytes of machine code that constitute the safety critical software in SSI is quite modest.

The validation effort that Short describes is rigorous and very thorough. The analysis has been aided by the fact that the SSI software is highly modular, and because the control flow is not complicated by the use of interrupts—polling mechanisms, as opposed to preemption mechanisms, have been used throughout. The analytic framework described includes functional, structural, information flow, and semantic analysis. These techniques have been applied in top down fashion through the modular structure of the software. Functional analysis checks the design against the (informal) requirements specification and identifies the requirements for each program module. Structural analysis checks the design and code for conformance to certain structured programming standards, and is intended to prove accessibility of every line of code. Information flow analysis detects illegal or omitted reference to variables. Given the control flow graph obtained by structural analysis, a semantic analysis assembles the individually validated modules into a validated whole, with a check that derived input/output relations correspond to the requirements. A detailed timing analysis is performed in the final review stage, prior to extensive online testing.

It seems that a completely formal treatment of the design path from high level system requirements to detailed timing analysis of the SSI generic program would present a major engineering challenge if conducted in a formal manner. Interesting though it would be to conduct a reappraisal of the correctness of the SSI software given the current state of the art, it is not what this thesis sets out to achieve (although see Chapter 6). Instead we consider an issue not mentioned by Short, nor even by Cribbens [24]—namely, the validation of the Geographic Data. Cribbens hints at the need for a "knowledge based approach to scheme design", but it was only later that proposals for formally based tools for Geographic Data preparation and analysis emerged [25]. The work reported in this thesis started from early consultations with Mitchell [66], but has progressed independently of British Rail's own research [48].

1.3.3 Examples of Geographic Data

A more thorough account of the Geographic Data Language is given in Chapter 2, but it is easy to introduce the main concepts occupying later chapters through a few examples. Figure 1.3 reproduces the ***scheme plan*** for the layout in Figure 1.1 with further annotations to show routes and ***sub-routes***. Route R_{28} proceeds from S_2 to S_8 through the points P_2 and P_3 reverse and normal respectively. In this scheme plan,

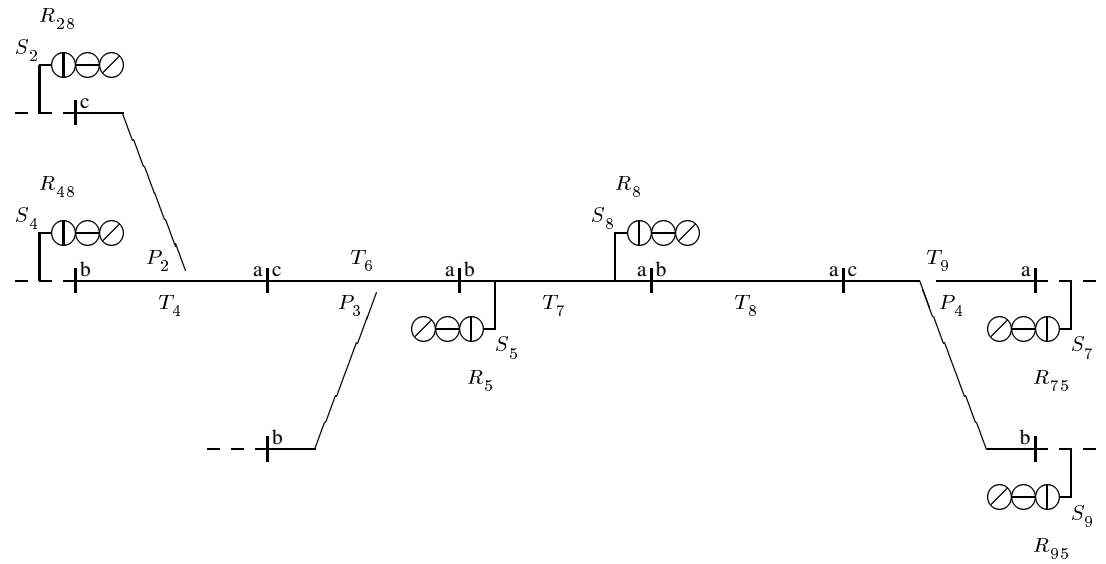


Figure 1.3: The scheme plan signalling layout in Figure 1.1 with route and sub-route annotations. Routes are identified paths between main signals, and each track circuit is associated with a collection of sub-routes so that a sub-route is defined for each path through a track circuit that lies on a route. A sub-route may be a component of more than one route (as T_7^{ba}, for instance).

there are four sub-routes associated with T_4: westward T_4^{ab} and T_4^{ac}, and eastward T_4^{ba} and T_4^{ca}. Thus R_{48} (say) can be identified with the sub-routes T_4^{ba}, T_6^{ca}, and T_7^{ba} in that order, by the points P_2 and P_3 which are required normal (and any ***overlap*** beyond the exit signal S_8, but shall not consider overlaps at present). These entities are control variables upon which the Geographic Data and control interpreter operate.

The Geographic Data are conceptually organised into a number of files each of which holds data that serves a specific purpose. Some of these files are accessed at random (as, for example, when a panel request is processed), whilst others are processed in rotation, once a major cycle. Thus, data in the ***input data file*** are responsible for copying the incoming status information to memory, and the ***output data file*** contains data that determine the command to be issued to each TFM as the system evolves. These data are accessed periodically, and there is one block of code to execute corresponding to each telegram. For example, the data listed for the command telegram for signal S_2 will specify the conditions under which the signal can be switched off (*i.e.*, from red to a less restrictive aspect). These will typically include checks that S_4, S_7 and S_9 are on, that the points on the route are detected (in some position), and that the track circuits to the next signal are clear. These data are designed to ensure that signals remain at red unless an onward route is locked (*e.g.*, by testing the appropriate route variables)—though this is a *property* that should be checked.

The conditions under which a route may be locked, and the locking conditions for the route (*i.e.*, the conditions that must not change while the route is set), are specified

by ***route request data***. For the running example:

$$\texttt{*Q28 if } P_2\,\texttt{crf}, P_3\,\texttt{cnf}, T_4^{ac}\,\texttt{f}, T_7^{ab}\,\texttt{f}$$
$$\texttt{then } R_{28}\,\texttt{s}, P_2\,\texttt{cr}, P_3\,\texttt{cn}, T_4^{ca}\,\texttt{l}, T_6^{ca}\,\texttt{l}, T_7^{ba}\,\texttt{l } \texttt{\textbackslash .}$$

This guarded command is a statement in the Geographic Data Language that is executed in response to a route request issued at the signal control panel. Execution begins at the *label* `*Q28` (which is treated as a pointer into the static data table), and continues without interruption up to the "`.`" which terminates the command. Several variables are tested here: points P_2 are tested to see if they are controlled reverse or "free to move" reverse (P_2 `crf`); similarly, P_3 `cnf` is a test to see if these points are normal or "free to move" normal (a more detailed discussion of the points test is deferred until Chapter 2). In addition, several conflicting sub-routes are tested (T_4^{ac} `f`, T_7^{ab} `f`) to check that they are *free*. If all these conditions are satisfied the route is locked by updating the variables as specified in the conclusion of the rule: the route variable is *set*, the points are *controlled reverse* and *controlled normal*, and the sub-routes are *locked*. (The terminology of railway signalling is used here, but it is mildly confusing to speak of the route being 'locked' by this action, rather than 'set', although the control variable for the route *is* 'set'. Note that the signalling actions in setting a route are firstly that it is 'locked', then it is 'proved'; the route is finally 'set' when the entrance signal displays a proceed aspect, usually green.)

Another class of Geographic Data specifies conditions that govern route release. It is (usually) necessary to lock routes in a single action, but they can be released gradually as the train proceeds, 'freeing' the network to the rear. Such bookkeeping is carried out by commands listed in the ***sub-route release*** data file which are executed sequentially over the course of a major cycle. Continuing with the example in Figure 1.3, these data may specify:

$$T_4^{ca}\,\texttt{f if } R_{28}\,\texttt{xs}, T_4\,\texttt{c } \texttt{\textbackslash .}$$
$$T_6^{ca}\,\texttt{f if } T_4^{ba}\,\texttt{f}, T_4^{ca}\,\texttt{f}, T_6\,\texttt{c } \texttt{\textbackslash .}$$
$$T_7^{ba}\,\texttt{f if } T_6^{ca}\,\texttt{f}, T_6^{ba}\,\texttt{f}, T_7\,\texttt{c } \texttt{\textbackslash .}$$

These rules introduce the following signalling principle: the first sub-route on a route can be released (freed) as soon as the route has been unset as long as the track circuit is clear; subsequent sub-routes are released in the sequence they are traversed. The sub-route release data must specify the sequence correctly. The order in which the rules are specified in the file, and hence the order in which they are executed, is immaterial—more precisely, safety properties of the interlocking must not depend on the order.

To illustrate the importance of this, the test T_7^{ab} `f` should be sufficient to guarantee that neither of the conflicting routes that terminate at S_5 is locked in when R_{28} is locked; indeed, if a train on R_{95} (say) has passed the entrance signal but not yet cleared

T_9, this test in the availability conditions for R_{28}, and similar tests in the rule for R_{95}, will ensure that these two routes do not interfere however far the train has progressed from S_9. The command to *unset* R_{28} is executed from the output data file (usually when the data for signal module S_2 are processed). A route is unset in response to a cancellation request from the signal control panel (or automatically as the train enters the route), but the conditions under which the entrance signal can be returned to red will depend on whether an approaching train is within sighting distance of the signal.

The problem we have to face is to determine whether the locking conditions in rules such as the above are adequate to ensure that trains do not run an undue risk of a collision or derailment. This is clearly not a trivial matter. In order to approach this subject the semantics of the Geographic Data Language are discussed further in Chapter 2, and properties of the data are examined in succeeding chapters. The next section introduces the ***remote route request*** protocol which is investigated in Chapter 6, and explains the mechanisms that enable several Interlockings at the control centre to communicate to achieve their collective management of larger railway networks.

1.4 Inter-SSI Communications

In any signalling scheme there may be a requirement, depending on the physical extent of the network, to divide the railway into a number of areas (or blocks), each controlled by a separate interlocking. Where SSI is concerned this distribution of control is further necessitated by the limited capacity of a single central interlocking processor. Limited capacity means the signalling area under the control of one operator will be divided between a number of Interlockings. On this scale the divisions may be rather small so it is important that boundaries are not only transparent to traffic in the network, but also transparent to the signal operator. The less fragmented the operator's view of the network is the better SSI can approach the broad aim in railway signalling of relieving the signal operator of the greater part of the burden of the safety of railway traffic.

In order for the control of a train to pass safely between interlocking areas some communication mechanism is needed to transfer information that needs to be shared about the status of the network in the fringe area. A typical situation is illustrated by the scheme plan in Figure 1.4 which focuses the discussion below. Here the cross-boundary routes converge before the boundary and terminate at a common exit signal. It is also possible that routes will diverge again after the boundary. In general there will be numerous lines linking the two interlockings. Signal engineering practice seeks to avoid placing boundaries through points since the complications introduced significantly increase the communication overheads. For the same reason boundaries are avoided if there would be points immediately beyond the signal at the boundary.

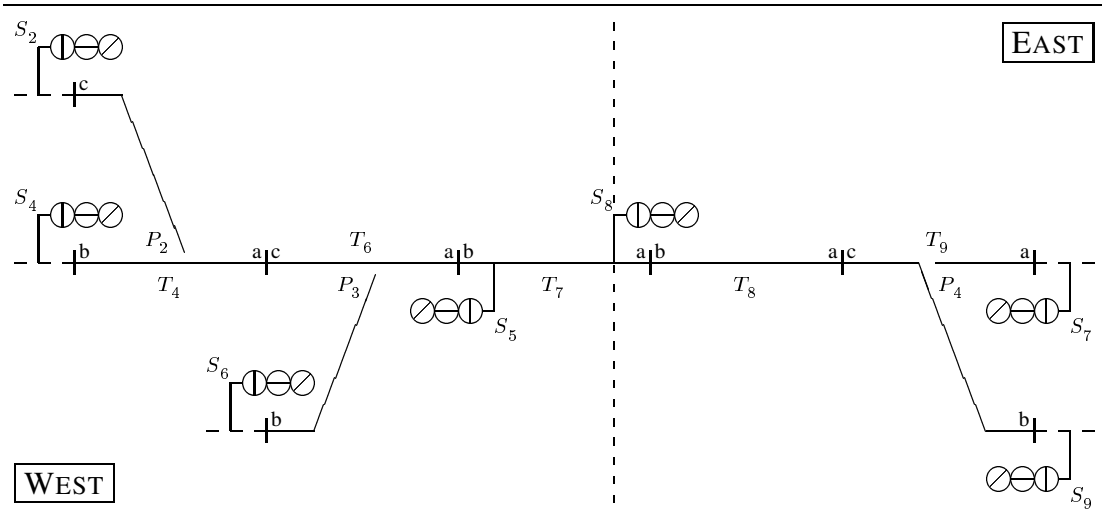


Figure 1.4: EAST and WEST communicate to set routes from entry signal S_7 or S_9 in EAST, to the exit signal (S_5) in WEST—since WEST controls the tail portion of both routes (just that over T_7, plus overlaps). There are no WEST to EAST routes as those up to S_8 are contained in WEST, and routes onward from this signal are controlled by EAST, as is the signal itself.

Data are transmitted between Interlockings by means of a high speed communication bus called the ***Internal Data Link***. Several Interlockings can be connected to a single bus, but normally an individual need only exchange data with its nearest neighbours. Outgoing ***IDL telegrams*** are prepared by commands in the Geographic Data and the generic control program is configured to copy their contents to the link at least once a major cycle. Two main classes of data are required to be communicated: continuously required data such as the aspects displayed by signals in advance of the boundary, and intermittently required data such as requests from one SSI for another to perform some signalling function such as moving a set of points or setting a route. Exactly what data need to be communicated depends on the nature of the boundary—our concern in Chapter 6 will only be with the complex situation of setting routes that are divided by Interlocking boundaries. Typically, the inter-SSI communications these induce occupy about twenty percent of the capacity of one Interlocking.

1.4.1 Setting Routes over Boundaries

Suppose that one wished to route a train from S_7 to S_5. On receiving the panel request for this route EAST first evaluates the availability conditions in its portion of the network: if these are not met the request simply fails, otherwise EAST must wait until it is certain the route is also available in the other Interlocking before locking it. To achieve this EAST issues a *remote route request* to WEST over the internal data link.

On receiving such an input WEST should handle the request just as it would handle route requests coming directly from the control panel—this simplifies the design of

the control interpreter, and data preparation. Thus an incoming IDL request will be translated into a panel request and queued in the usual manner. When a remote route request is subsequently processed the difference is that WEST must communicate to EAST if, or when, the route is locked: WEST sends its acknowledgement via a reply telegram to EAST over the IDL.

In EAST the acknowledgement is also treated as a remote route request: on this occasion EAST proceeds to lock the route it had originally requested. The two Interlockings need to use a dedicated pair of IDL telegrams to communicate request codes and their acknowledgements. Normally, many routes over numerous lines link the two signalling areas, but a single pair of (eight bit) telegrams should suffice to carry all the necessary request codes and their acknowledgements. To summarise:

1. EAST receives a panel route request for a cross-boundary route. If the route is available in EAST, issue a remote route request to WEST.

2. WEST receives an IDL input conveying a remote route request. If the route is available, lock the route and reply to EAST with an acknowledge telegram.

3. EAST receives a reply telegram to the earlier remote route request: it can then lock the route and control the entrance signal as usual.

Once the route has been locked in EAST the aspect of the entrance signal can be changed if the prevailing conditions allow this. For example, only if the tracks down to the exit signal are clear, and if opposing signals are on, can EAST clear the signal—to green or yellow, depending on the aspect displayed by the exit signal. Thus, in addition to the telegrams used to convey request codes, another IDL telegram is needed to convey the status of tracks and signals in the fringe area. Such data are needed continuously.

1.4.2 Releasing Sub-routes over Boundaries

Once a train has passed the entrance signal and progressed along the route (or if the route is subsequently cancelled by the signal operator) the sub-routes along it should be released in the usual manner—*i.e.*, by rules in the continuously executed sub-route release data (as in Section 1.3.3). At least, the sub-routes can be freed in this way up to the boundary: T_8^{ab} is a control variable in EAST of course, while T_7^{ab} is in WEST, so the usual rule for freeing the 'inward' sub-route does not apply.

In order for the whole of the route to become free EAST must send a request to WEST for it to release its part of the route once the correct circumstances obtain. If the sub-route release mechanism is to be transparent (to the operators) the necessary *cancellation request* should be issued automatically. To achieve this in SSI the correct

circumstances are recognised by rules in the sub-route release data. If WEST receives a cancellation request it can release the inward portion of the cross-boundary route unconditionally. Furthermore, WEST should acknowledge the cancellation request so that EAST will be aware that the route has indeed been released. The usual sub-route release mechanism in WEST will ensure that the remainder of the route is released as the train proceeds to the next signal. To summarise:

4. Whenever conditions indicate that a route has cleared up to the boundary, EAST issues a remote cancellation request to WEST.

5. When WEST receives a request to cancel an inward route it does so unconditionally, and acknowledges the request with a reply telegram to EAST.

6. On receipt of such an acknowledgement, EAST should cease to issue cancellation requests, the route having been cancelled in both Interlockings.

1.4.3 Implementing Remote Route Locking

With the current generation of Solid State Interlocking the number of IDL telegrams that can be used is limited to a maximum of fifteen in total. Each IDL telegram conveys eight data bits, and the Interlockings connected to the link take it in turns to transmit all fifteen bytes of data in a round-robin protocol: the transport layer is configured so that each SSI broadcasts its data at least once a major cycle (the frequency depends on the number of Interlockings connected to the link). On receipt of an IDL data packet the SSI is able to extract those bytes that are relevant to it (this address information can be computed statically, and is 'burned' into EPROM when the system is installed). Since the outgoing IDL telegram will be written at arbitrary times during a major cycle it is necessary to buffer the telegrams. As a consequence the protocol as presented is far from being robust as the various uses of the request telegram can interfere with one another. If one SSI locks the inward portion of a route in response to a remote route request, the (buffered) reply telegram should not be overwritten before it can be sent. While not unsafe, in extreme circumstances this may lead to livelock, and other problems. Another reason why the protocol sketched above is not correct is that the remote route request may simply fail in the second Interlocking (WEST), but the first (EAST) has to be notified of this failure.

Such concerns introduce the need for telegram protection and timers. To implement remote route locking the designer has access to a collection of ***elapsed timers*** which may be stopped and started by commands from the Geographic Data, but which are otherwise updated by the (real-time) generic program. Note that an elapsed timer can serve both purposes if we can differentiate between a timer that is *stopped*, and one

that is *running*. One timer is needed for each IDL telegram used to convey request codes to another SSI, but other control data are needed to implement the sub-route release mechanism over the boundary. The details are drawn out in Chapter 6 where safety properties of these inter-SSI communications will be examined. Until then our concern will be with the safety properties of the Geographic Data within a single SSI.

1.5 Formal Approaches to Signalling Safety

In seeking to adopt rigorous techniques British Rail (now Railtrack), along with several other railway authorities, advance the opinion that the more formal analysis will improve the quality and safety of their products and services. While the reasons for introducing computer controlled railway signalling may be largely economic, it seems that with the advent of design notations such as the Geographic Data Language the overall safety case can be strengthened because of the possibility to produce formal proofs of the behaviour of the interlocking. Even without formal proofs the possibility to test (simulate) the design long before tracks are laid down, and with full confidence that the same software will control the live network, is a considerable boost to safety and productivity. In principle at least, the introduction of more rigorous techniques will improve productivity in the long run because a formal proof that the Geographic Data are safe may remove much of the need for testing the design.

These arguments indicate that what is required is a framework within which to conduct various forms of analysis on Geographic Data. Simulation and testing remain central concerns in signalling engineering—if only because of the need to test the final data/control configuration for each instantiation of the data. In the context of this thesis the 'formal correctness' of the Solid State Interlocking is not in itself the issue. Rather, the central problem is that of automatically checking SSI data through an appropriate language of logical assertions and proof. There appear to be two distinct approaches to providing the analytic framework required: either we attempt to formalise the principles of railway signalling, or we reduce a given design to a formal specification (or model) whose properties we verify. A brief survey of related work will help to illustrate these choices.

1.5.1 Related Work

The treatise by O. S. Nock [76] sets out in considerable detail the recent signalling engineering practice on British railways. Nock deals both with the *static* and *dynamic* issues of scheme design: static issues include network topology, the placement of signals, and their relative separation; dynamic issues address the relationships between train positions and signal aspects, and the separation between trains, *etc.*, in the evol-

ution of the network topology (as points settings change), and as trains proceed. For example, Nock can be interpreted to yield the following requirements for clearing the entrance signal of an uncomplicated (mainline) route without overlaps:

i All track circuits on the route must be *clear*;

ii Track circuits on all conflicting routes up to the point of conflict must be *clear*;

iii All points on the route must be *controlled* and *detected* (in correct position);

iv The exit signal must be *alight* (*i.e.*, drawing current);

v The entry signal for all conflicting routes must be *on*.

The route's entrance signal may go *off* if and only if these conditions are satisfied, and the route is locked. One persistent problem for signalling engineers is to decide whether all conflicting routes have been identified.

Recent work by King [51] records the current signalling rules applied by Railtrack. King's layered Z specification is intended to form part of requirements specification documents used for the procurement of signalling systems. The first layer defines the concept of network topology in terms of primitive track components (points, plain track and ***diamond crossings***) and their allowable interconnections. Paths, and the concept of interference between paths, are defined on this static component. The second layer formalises the dynamic signalling rules in a *conceptual foundation*—since this is supposed to be independent of any particular technology, trains themselves are modelled (in terms of the paths they are on and in which they may come to a halt).

The third layer in King's specification is an instantiation of the conceptual foundation (*e.g.*, SSI, which introduces signals, routes and sub-routes as control elements). However, since King does not address the question of whether the rules formalised in the conceptual foundation are 'safe' (or at least consistent), it is inevitable, as the author himself points out, that verification of the safety properties of any *formal refinement* of the conceptual foundation will be needed. The Z specification cannot therefore be used to define safety requirements—although safety can of course be defined in terms of the conceptual foundation.

Wong [99] also attempts to codify the dynamic signalling rules in a formal theory. He proposes a scheme design methodology that links a theorem prover for higher-order logic with CAD tools for signalling scheme plans. The theorem prover automatically checks that the network described is legal with respect to some simple rules for assembling network components (*e.g.*, that it forms a finite connected graph). Wong goes on to generate ***control tables*** for each route inferred from the scheme plan—these are specification documents used to guide the preparation of Geographic Data for an SSI

installation. This is practically interesting because of the difficulty of certifying that the route setting conditions specified in the control table are sufficient (for safety).

However, Wong does not use his HOL theory directly to address the question of the adequacy of control table route specifications. Instead he derives the behaviour of the railway from the network structure, presenting the model as a finite state automaton. In higher-order logic a 'time' varying function describes the state of the network at any instant, and the behaviour is governed by the dynamic signalling rules. Wong demonstrates how to prove that the automaton is safe with respect to the property that no two conflicting routes can be simultaneously set. Unhappily Wong's demonstration is somewhat vacuous since the notion of *conflict* in defining the formal property is identical to that used to encode Nock's requirements. Cullyer and Wong [26] have used a similar model to examine safety related properties of a level crossing.

Some related work on behalf of the Danish State Railways has been carried out by Mark Hansen [56] under the aegis of the ProCos project. Her VDM specification is also based on a description of the network topology, and the purpose of the model is to clarify formal requirements (functional, as well as safety) for interlockings to be developed on a per station basis. This work emphasises model validation (through simulation), and requirements capture. The principal requirement is that trains do not collide. Modulo the usual caveats about coupling trains, this is expressed in a predicate that asserts that no track section contains more than one train. An attribute of track sections in the model is, therefore, that they may be associated with a *set* of trains. The hidden assumption here is that track-side equipment is capable of determining that a second or subsequent train has entered an already occupied track section. Track circuits are unable to decide this, for example, although more sophisticated train detection systems can relay train identities to the control system (trains and tracks communicate).

The authors cited above record the (static and) dynamic signalling rules in a mathematical theory. The natural focus in these enterprises is on requirements capture, with safety requirements dominating. But this begs the question of how to demonstrate that a purported implementation conforms to the requirements—in particular, to verify that the interlocking is safe. Another body of work addresses the verification problem directly.

Atkinson and Cunningham [4] describe a signalling case-study that exercised a tableaux proof system for Modal Action Logic (a variant of PDL). This took a simple interlocking (the FOREST LOOP scheme [8, illustrated on page 215]) described by a system of MAL axioms derived from the Geographic Data and the network topology, and further *action* rules to describe permitted train movements. The idea is to prove that a modal property p is a consequence of such a specification $Spec$: the MAL prover

attempts to refute the goal $Spec \Rightarrow \neg p$. The logic has a refutation complete decision procedure to prove such goals—meaning that if a counter model exists it will always be found. The procedure is semi-decidable however, so cannot always prove $Spec \Rightarrow p$ when it is true. This case-study can fairly be said to demonstrate the capabilities of the FOREST tools, but the model is too concrete to be a *specification*, and too impoverished (in its notion of time and computation step in calculating signal aspects, for example) to be useful as a vehicle for proving safety of the interlocking.

In a similar, but much more successful vein, Stlmarck and Säflund model interlockings for the Swedish railway authorities (Banverket, and SJ, the railway company) using propositional logic [88]. They have developed dedicated tools for analysing safety related properties of STERNOL programs which are used in interlocking design. A STERNOL program is a system of equations, with one equation—really, a guarded command—for each value a variable in the program can take. One group of equations may refer, for example, to the aspect of a particular signal. The Circuit Verification Tool [88] is used to verify that exactly one of the guards in the equations for a program variable is true at any time. This guarantees determinism (in each execution cycle), and such a STERNOL program can therefore be implemented by executing the equations in *any* order (cyclically). For SJ and their subcontractors this is an important safety property of their interlockings.

By representing a STERNOL program in propositional logic it is possible to go on to examine other safety properties by proving $Prog \Rightarrow p$. Given the size of such formulas, this would be a severe challenge but for Stlmarck's (patented) natural deduction style proof technique for Boolean satisfiability. The time complexity of the algorithm is polynomially related to the number of subterms in the formula, with the exponent (hardness) being determined by the number of simultaneous free assumptions needed in the natural deduction proof tree. The empirical evidence is that for many practical problems the degree of hardness is low by this measure, so the proof technique is effective for extremely large formulae (exceeding 10^5 connectives). To obtain counter models an ordering on the subterms in the formula is needed; this does not affect the hardness of the proof, but makes space requirements (*i.e.*, the length of the proof) sensitive to the order chosen.

The Vital Processor Interlocking (VPI) analysed by Groote *et al.* [37] for the Dutch Railway Company has an execution model that is similar to the STERNOL programs described above: a sequence of equations that are solved once a cycle, each of which defines the value of an internal control variable or system output. Groote and his colleagues formulate safety requirements in a modal logic so as to express properties relating to finite sequences of program steps (or states): *next* formulae refer to a finite future, *just* formulae refer to finite immediate past, and *static* formulae refer only to

the current state. To prove that the program satisfies a modal property both are translated into propositional logic, and $Prog \Rightarrow p$ is checked using Stlmarck's method. In general, if the property refers to n time steps then this many time-indexed copies of the program are needed for the proof. Clearly n and the size of $Prog$, in terms of the number of subformulae, determine the range of application of this approach to VPI program verification. Taken literally, n may be 'large' (21 s in the example), but a time abstraction alleviates the complexity problem so that the results are encouraging.

Returning, finally, to the problem of verifying safety properties of Geographic Data, we should note British Rail's own research which tackles the problem from an automata-theoretic perspective. Ingleby and Mitchell [48] represent SSI behaviour in a Mealy machine having next state and output functions ν and ω. Safety properties are characterised as state predicates (*static* properties in Groote's terminology [37]) and output predicates (also static). The problem is to demonstrate that the automaton modelling the interlocking is *safety transitive*: if $safe(s)$ asserts that state s is safe, the interlocking is safety transitive if for all states s and inputs i, $safe(s) \Rightarrow safe(\nu(s,i))$. This proof concept [74, illustrated more coherently in Chapter 4] is an instance of a powerful proof technique for safety properties called *co-induction*.

Ingleby's data decompositions [47] (discussed further in Chapter 5) are what makes this approach to automatic verification of safety properties of Geographic Data practical at all: these lead to state clustering in the automaton, and a *local* proof strategy, but complicate the task of generating counter models and tracing the location in the data where the safety properties are being violated. Related work has been reported by Conroy and Pulley [21] whose models are Büchi automata. These authors are plagued by the enormous combinatorial complexity of the reachable state space in signal interlockings. For the Hoorn-Kersenboogerd interlocking [37], Groote puts the reachable state space somewhere between 10^{30} and 10^{500} states: the lower bound arises because the VPI records circa 100 Boolean inputs—SSI receives up to 512 such inputs in each major cycle (ignoring the occasional panel request) and the internal state is a vector of (at most) 1,216 bytes.

1.5.2 Contributions & Thesis Overview

When approaching the question of automated verification, the cited work illustrates that great care should be exercised to avoid intractable state spaces on the one hand, and combinatorial explosion in checking Boolean formulae on the other. Both can be avoided by selecting an appropriate abstraction with which to work, and as long as safety can be satisfactorily formalised through invariants of the internal state of the SSI (which may still refer to system inputs and outputs of course). We concentrate on the semantics of the Geographic Data Language which, in programming terms,

is a somewhat richer language than STERNOL or the Vital Logic Code in VPI. The focus on semantics leads to a *compositional* verification strategy—but the route to this understanding of the problem is as important to record here as the final synopsis itself.

Chapter 2 The next chapter fills out the background, concentrating on the syntax and semantics of the Geographic Data Language, and explains the overall organisation of the data in the SSI. The Geographic Data Preparation Guide [9] defines the language informally, in a manner that is intelligible to signalling engineers, but to prove properties of programs written in the language a more rigorous understanding is needed. In this chapter therefore, a formal semantics is proposed that is faithful to the informal description. This defines the execution model that is elsewhere assumed to be valid.

Chapter 3 The starting point was not a formal description of the Geographic Data Language, but a model of a much simplified signalling scheme. The model is derived by a systematic translation of the data into CCS. The execution model abstracts from such details as minor and major cycles, concentrating only on the transitions allowed by the rules held in the database. The focus then is on the *properties* of the Geographic Data, and their formulation in terms of a predicate **F** of the states of the abstract machine M. For the simple example we can verify these properties by model checking: we prove **F** is invariant, that is $M \models \nu Z.\mathbf{F} \wedge [-]Z$ in the terminology of the modal μ-calculus, using the Concurrency Workbench and tools developed for the task. However, this verification method does not scale beyond the small examples tried—the problem is one of abstraction in the *proof*.

Chapter 4 This chapter examines the invariance proof in detail. Instead of trying to establish that (all reachable) states of the model are safe we prove that the transitions *preserve safety*. This gives a much more direct demonstration that the Geographic Data are safe because the data really define the state transitions of the SSI (with respect to the semantics). The (co-)inductive nature of the proof is explained here in terms of the proof tableau constructed by the model checker used in Chapter 3: the idea is to show that if a state S is safe (*i.e.*, $S \models \mathbf{F}$) then every state S' that is immediately reachable from S is safe: $S' \models \mathbf{F}$. We do not worry about whether S is reachable. Since the proof method is to be mechanised, time is taken here to identify the steps needed to demonstrate that these *verification conditions* are true.

Chapter 5 The arguments formulated in Chapter 4 are interpreted here in the framework of Floyd-Hoare logic. In this chapter the syntax and semantics of the Geographic Data Language are formalised as a theory of higher-order logic, and embedded in the HOL proof system. From this theory the program logic is derived, and the proof obligation formulated in the goal $\{\mathbf{F}\}\, c\, \{\mathbf{F}\}$, for each command c in the data. Through the

tactics of the HOL system, and a modest amount of ML programming, we recover a fully automatic proof method which is linear in both in the length of **F** and the number of commands c (and independent of the number of states of the SSI). We show that the range of application of this prototype Geographic Data verifier can be considerably extended through techniques for decomposing the invariance proofs. Decomposition according to the structure of c comes for free with Floyd-Hoare logic, so we concentrate on decomposing **F**.

Chapter 6 Here the model presented in Chapter 3 is developed in a different way to analyse properties of the inter-SSI communications—specifically, those by which two Interlockings cooperate in locking routes over their common boundary. The logic to achieve this route locking (and release) is also encoded in Geographic Data. It is found that unfavourable message delays can lead to circumstances in which hazards that compromise the safety of railway traffic can arise *in principle*. Since such hazards are not precluded *in practice*, a strict interpretation of the term 'safety' leads to the conclusion that this is a design flaw in the remote route request protocol. In fact the risk implied by this fault in the generic program is difficult to quantify precisely—which is sufficient reason to study the problem formally. Our analysis leads to several recommendations to eliminate the flaw, and we prove that it is possible to implement the protocol so that it cannot then, of itself, lead to unsafe states in the railway.

Finally, **Chapter 7** concludes this work with a summary, and indicates the likely impact of our findings on the industrial usage of formal methods and the practice of interlocking design. The application of the theorem prover developed in Chapter 5 to 'live' data from the Leamington Spa signalling scheme is described in this final chapter. Also considered are the concrete recommendations coming from the analysis in Chapter 6: these indicate that only very minor changes to the SSI generic program would be needed to address the concerns raised there.

Chapter 2

The Geographic Data Language

It is the main purpose of this thesis to devise an approach to the verification of safety properties of Geographic Data. The Introduction described the relevant features of Solid State Interlocking to provide the necessary context. The focus in this chapter will be on the language in which the interlocking functions are encoded. Section 2.2 explains the overall organisation of the Geographic Data which are conceptually arranged in a number of files at the source level. In Section 2.3 the (concrete) syntax of the Geographic Data Language is given, accompanied by an explanation of the intuitive meaning of the various constructs. Sections 2.4 and 2.5 introduce rigour to the underlying execution model by providing the language with a mathematically precise semantics.

2.1 Introduction

Due to the undesirability of developing and verifying the correct implementation of a separate control program for each SSI installation, the system's software has been separated into its *control* and *data* parts. The control part is independent of the specific signalling functions and is implemented in the generic software which is the same in every installation. The Geographic Data Language expresses the specific signalling functions which vary from Interlocking to Interlocking. This is an application specific language designed to be intelligible to railway signalling engineers without their needing specialised knowledge of computer programming. The SSI generic program *interprets* these data, and for this reason is usually referred to as the *control interpreter* in the sequel. In fact, the program interprets a (byte) compiled version of the Geographic Data, so the correct implementation of the compiler is also an issue that should be addressed in checking (safety) properties of the data.

This separation of concerns means one can use very different techniques to validate the generic software on the one hand, and the Geographic Data on the other. Whereas the control interpreter requires to be validated with respect to its require-

ments only once (see Section 1.3.2), the Geographic Data have to validated with respect to the prevailing principles of signal engineering at each installation. We expect the (safety) properties of the generic software to be independent of the data interpreted, although precise timing properties inevitably depend on the final data/control configuration. There are also some (functional) properties of the combination that cannot be verified by considering these aspects of the SSI software in isolation. The principal culprit in this respect is the remote route request protocol examined in Chapter 6.

Properties of the Geographic Data, however, only depend on the execution model supplied by the control interpreter. It will therefore be fruitful to formalise the semantics of the Geographic Data Language. On the one hand this provides a reference for the language against which we can judge whether the compiler and the interpreter have been correctly implemented, and on the other it provides a precise mathematical framework in which to conduct *proofs* about the behaviour of the interlocking. These semantics are discussed in Sections 2.4 and 2.5 below. This focus leads to the treatment of the Geographic Data as a *program* which has static and dynamic components. The static data are the rules listed in the database—these are code fragments stored in EPROM, and are what is meant when referring to *the* Geographic Data in the sequel. The dynamic component is the memory on which the data and generic program operate.

2.2 Static Data and Dynamic Data

At the source level the data are separated into a number of files that deal with distinct interlocking functions. The static data can be broadly placed into two groups: those data that are executed periodically over the course of a major cycle, and those that are accessed randomly. Concrete examples are given in the next section—here we are interested in the data's overall organisation, and the general functions they are to perform. The glossary in Appendix A.3 accompanies this section.

2.2.1 Geographic Data Identity Files

The dynamic component of a 'Geographic Data program' is given by a collection of state variables upon which the static data operate—these constitute the internal state of the SSI, stored in RAM. One variable is defined for each physical control device—*i.e.*, for each signal, track circuit and point switch—as well as for each logical control. Logical controls include routes, ***sub-routes*** and ***sub-overlaps***, timers and latches (as well as the telegrams used to communicate with external devices). These variables are globally declared and may be accessed throughout the Geographic Data. *Identity files* define sets of variables of appropriate types:

TCS the status of a track circuit may be *undefined*, *occupied* or *clear*. An eight-bit timer in ***track circuit memory*** records how long it has been in the current state (up to 254 seconds).

PTS each point switch is represented by two four-bit records, one for the *normal*, and one for the *reverse* lie of the switch. The normal or reverse field must be specified whenever ***points memory*** is accessed in the data.

ROU each route through the network is represented by two control bits. Routes may be *set* or *unset*. Routes may also be barred by clearing the other control bit, but this can only be modified from the technician's console.

FLG flags are single bit control variables. In the sequel we are mainly concerned with sub-routes: these, and sub-overlaps, may be either *locked* or *free*.

SIG signals have many attributes and require three bytes of data. One byte is a timer, three bits indicate the aspect to display, and other fields are for control information such as the status of the lamp proving circuit, recording cancellation requests from the signal control panel, *etc.*.

In addition, each panel request is identified, and a collection of general purpose ***elapsed timers*** is provided (used to implement the remote route request protocol, and for ***swinging overlaps***). In the sequel we shall let the script letters $\mathcal{P}$, $\mathcal{R}$, $\mathcal{S}$, $\mathcal{T}$, and $\mathcal{U}$ stand for the sets of points, routes, signals, track circuits and sub-routes declared in the Interlocking; $\mathcal{Q}$ is the set of panel requests.

2.2.2 Source Files: Periodic Access

One major cycle is divided into 64 minor cycles irrespective of the actual number (≤ 63) of track-side modules with which the Interlocking communicates. One incoming data telegram is processed, and one outgoing command telegram is processed in each minor cycle. Command and data telegrams convey up to eight bits of information. A block of data is associated with each telegram, drawn from the appropriate Geographic Data *source file*:

IPT One block of data is associated with each input telegram received from the track-side functional modules. These data are normally very simple since all that is required is to copy the bit-fields in the incoming telegram to the internal state. In preparing these data (a sequence of assignments) the signalling engineer has to be careful to associate the input fields, which correspond to the physical output pins in the TFM, with the correct data variable—for instance, bits 7 (and 5) and 6 (and 4) conventionally refer to the *detected normal* and *detected reverse*

fields when the message is from a points module. Low-order bits are used for track circuit inputs from the device, which are likewise copied to track circuit memory.

OPT One block of data is needed for each TFM addressed by the Interlocking. When the command is to a points module these data are again rather simple since all that is required is to copy the *controlled normal* and *controlled reverse* fields in points memory to the appropriate outputs. More complex instructions are necessary when the command is to a signal module since it is these data that must calculate the correct aspect to be displayed. This calculation depends on the aspects of neighbouring signals, which if any of the onward routes is locked, and on the proximity of trains to the signal. Several other attributes of the signal (not transmitted in the command telegram) have also to be computed by the output telegram data. These calculations are repeated every major cycle.

FOP Each command in the flag operations data file is executed in sequence, once a major cycle. These data perform various bookkeeping functions, the most important of which is *sub-route release* (see Section 1.3.3). The flag operations data are essentially guarded commands, and the control interpreter will execute $1/64^{\text{th}}$ of these in each minor cycle.

During one major cycle therefore, all of the data in the *IPT*, *OPT*, and *FOP* data files will be executed once, and the data will be executed in the same order (the polling sequence, specified by the signalling engineer when the data are compiled) in every major cycle. The *IPT* and *OPT* data are listed in the same order, input telegram m corresponding to output telegram m (and the TFM with that index), but being processed in minor cycle $m + 1$ (modulo 64).

2.2.3 Source Files: Random Access

The other source files considered here contain geographic conditions and commands that only need to be accessed on demand:

PRR Each input from the signal control panel corresponds to a command, or command sequence. The *panel route request* data file lists all route requests that arrive from the panel processor (or, as described in Section 1.4, from another SSI), and all route cancellation requests. The block of code associated with a route request usually consists of a conditional statement that tests the internal state to ascertain if the route availability conditions are met, and a command sequence to update the internal state accordingly, locking the route. Subsequent processing of the *OPT* data will effect the necessary changes in the network to set the route.

PFM Points "free to move" data specify the conditions under which points may be switched, with one set of data required for each lie of the points (normal or reverse). The conditions are shared by instructions elsewhere in the data, notably in the *PRR* data since route availability always depends on being able to move the points on the route to the correct position. Several routes may pass through the same collection of switches. The *PFM* data therefore help to reduce the volume of instructions needed in the database; they simplify the data according to the geographic principle that the conditions for moving the points are local ones, and this, in turn, reduces the likelihood of introducing errors in the route specifications. The interpretation of the *PFM* data is discussed in Section 2.4.

MAP Map data typically define a partial graph of the railway network. Searches are performed on this graph whenever it is necessary to look for evidence of a train in the approach to a signal (*e.g.*, an occupied track section). Such searches are often performed as part of the aspect calculation in the *OPT* data for a signal module, and will be used to decide if a route can be cancelled. Unlike the *PFM* data, the *MAP* data do not specify static *tests* on the internal state: instead, when the control interpreter encounters a *map search* it executes an algorithm to dynamically compute the test to perform given a starting point in the graph, and a set of termination points. Map search data are also discussed in Section 2.4.

Once the control interpreter begins to execute a block of data from one of the above sources, it continues without interruption until the block has been completed. The control interpreter is a sequential program so this behaviour is expected for the periodically accessed data processed as part of the standard minor cycle. However, this is also the case for the data executed on demand, so it is vital to be able to predict timing bounds for the execution of each code fragment. The Geographic Data Language admits only assignment of constants to variables, branching and sequence (and a simple code sharing mechanism), permitting one to accurately predict upper timing bounds.

Occasionally it is found, by a timing analysis of the Geographic Data, that the code for a panel request cannot be computed inside the permitted 20 ms. In such cases the data must be split over two or more minor cycles. This is achieved by a mechanism to add a panel request to the input buffer via a data command—the control interpreter processes the first part of the data for such a panel request, and queues a second panel request for the continuation to be processed in a subsequent minor cycle. At most one panel request is processed each minor cycle. It seems, even from this informal description of events, that the practice of splitting panel requests may introduce unpredictable behaviour because of the possibility that conflicting requests may intervene—although predictability may be recovered by placing the subsequent parts

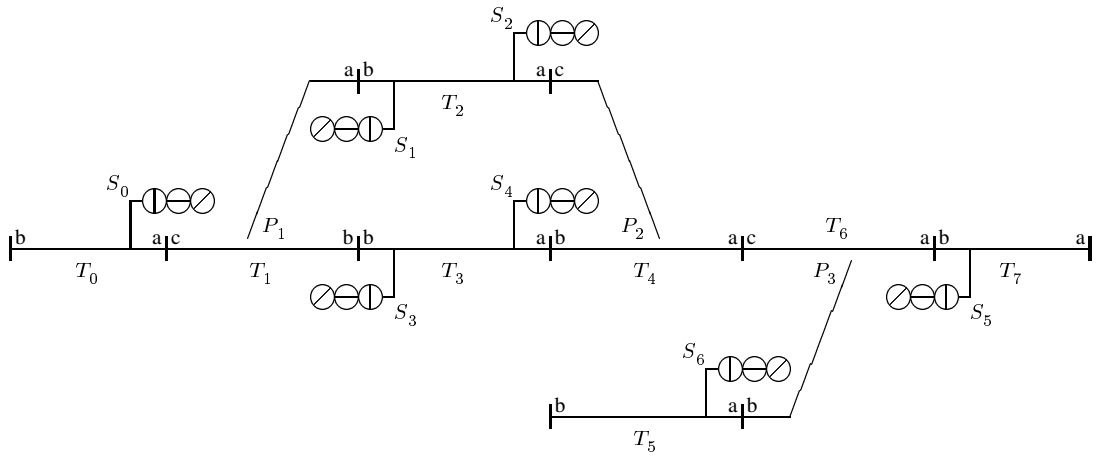


Figure 2.1: Signalling scheme plan for WEST

of the split panel request always at the *head* of the queue rather than at the tail, but the author does not know if the generic program has this behaviour. In any case, it is clear that validation of the Geographic Data is an issue that demands considerable effort. Presently the data are prepared by hand and (visually) inspected for errors by the engineers responsible for the design of the interlocking [25]. Software supports various kinds of syntactic analysis only (but several authors have begun to address the problem of providing semantics-based tools to support interlocking design and data validation [48, 75, 99, 88]).

2.3 Geographic Data Source File Syntax

This section spells out the details of the concrete syntax of the Geographic Data Language, although we shall restrict attention here, and in the rest of this thesis, to the route locking data in the *PRR*, *FOP*, and *PFM* (and *MAP*) data files. The semantics of the language will be clarified in Section 2.4, but we begin with a few examples drawn from the data for the signalling scheme in Figure 2.1. WEST will serve as a concrete example for this and subsequent chapters when such is needed. The entities declared in this scheme are:

$\mathcal{T}$	$\{T_0, T_1, \ldots, T_7\}$	Track Circuits
$\mathcal{P}$	$\{P_1, P_2, P_3\}$	Points
$\mathcal{S}$	$\{S_0, S_1, \ldots, S_6\}$	Signals

which represent physical entities in the network, and

$\mathcal{R}$	$\{R_{02}, R_{04}, R_1, R_2, R_3, R_4, R_5, R_{51}, R_{53}, R_6\}$	Routes
$\mathcal{U}$	$\{T_0^{ab}, T_0^{ba}, T_1^{ca}, \ldots, T_7^{ba}\}$	Sub-routes

which represent logical control entities. The set of panel requests $\mathcal{Q}$ is also declared: $\{\texttt{Q02}, \texttt{Q04}, \ldots, \texttt{Q6}, \ldots\}$.

2.3.1 Examples: Route Locking & Release

The conditions under which a route can be locked (prior to being set), and the locking conditions for the route (*i.e.*, the conditions that must not change while the route is set), are specified by data in the *PRR* and *PFM* files. For R_4 and R_{51} in WEST we have:

```
*Q4 if  P2 cnf , P3 cnf , T4ab f , T7ab f
        then  R4 s , P2 cn , P3 cn , T4ba l , T6ca l , T7ba l \ .

*Q51 if  P2 crf , P3 cnf , T2ba f , T6ca f
         then  R51 s , P2 cr , P3 cn , T6ac l , T4ac l , T2ab l \ .
```

It is not necessary to test all opposing sub-routes in the availability conditions for a route—thus `*Q4` does not check T_6^{ac}, for example. It *is* necessary to test T_7^{ab} in this rule because it is the last sub-route on all routes terminating at S_5 (but sub-routes further to the east do not need testing since it is required to release sub-routes in sequence). In the same rule, it is also necessary to test T_4^{ab} because this sub-route opposes the first sub-route on the route, and is the last conflicting sub-route on routes that also require the points P_2 normal (*i.e.*, R_{53}). Similar principles apply to `*Q51`, and all other main routes on British railways.

As a matter of principle, all points on the route should be checked in the availability conditions. For R_{51} the first set of points (P_3) are required in the normal position, and the second set are required reverse. The control interpreter evaluates P_2 `crf` as a disjunctive test; it first checks whether the points are already controlled reverse (P_2 `cr`) and if they are not evaluates the *PFM* data:

```
*P2N   T4ac f , T4ca f , T4 c \
*P2R   T4ab f , T4ba f , T4 c \
```

The points P_2 can be moved to the normal position if the reverse sub-routes are free, and the track circuit is clear. Conversely, the points are "free to move" reverse if the normal sub-routes are free and the track circuit is clear.

The *FOP* data specify route release conditions: it is (usually) necessary to lock a route in a single action, but routes can be released gradually as the train proceeds, and the tracks in the rear can be released and made available to other routes. Such bookkeeping is carried out by commands in the *FOP* data file which are executed sequentially over the course of a major cycle. These data may specify:

```
T6ac f if  R51 xs , R53 xs , T6 c \ .
T4ac f if  T6ac f , T4 c \ .
T2ab f if  T4ac f , T2 c \ .
```

$$
\begin{array}{rcl}
\langle \text{cs} \rangle & ::= & \texttt{if} \left[\langle \text{gc} \rangle \right]^* \texttt{then} \left[\langle \text{oc} \rangle \right]^* \left[\texttt{else} \left[\langle \text{oc} \rangle \right]^* \right] \backslash \\
\langle \text{gc} \rangle & ::= & \langle \text{test} \rangle \\
& | & \langle \text{map} \rangle \\
& | & \texttt{(} \left[\langle \text{gc} \rangle \right]^+ \left[\texttt{or} \left[\langle \text{gc} \rangle \right]^+ \right]^* \texttt{)} \\
& | & \texttt{@L} \\
\langle \text{oc} \rangle & ::= & \langle \text{cmd} \rangle \\
& | & \langle \text{cs} \rangle \\
& | & \langle \text{sc} \rangle \\
& | & \texttt{@L} \\
\langle \text{sc} \rangle & ::= & \texttt{(} \langle \text{ac} \rangle \left[\texttt{or} \langle \text{ac} \rangle \right]^* \texttt{or} \left[\langle \text{oc} \rangle \right]^+ \texttt{)} \\
\langle \text{ac} \rangle & ::= & \texttt{if} \left[\langle \text{gc} \rangle \right]^* \texttt{then} \left[\langle \text{oc} \rangle \right]^* \backslash \\
\langle \text{ev_set} \rangle & ::= & \texttt{if} \left[\langle \text{gc} \rangle \right]^* \backslash \\
\langle \text{ex_set} \rangle & ::= & \left[\langle \text{oc} \rangle \right]^* \backslash
\end{array}
$$

Figure 2.2: Geographic Data: conditional language constructs

The first sub-route on a route is released (freed) as soon as the route has been unset as long as the track circuit is clear; subsequent sub-routes are released in the sequence traversed. Note that the *PFM* and the sub-route release data (in particular) are specified in accordance with the geographic principle: the interlocking of elements in the railway depend on *local* components only.

2.3.2 Concrete Syntax of the Geographic Data Language

The syntax of the conditional language used throughout the Geographic Data is given by the grammar displayed in Figure 2.2. A conditional statement $\langle \text{cs} \rangle$ contains a list of geographic conditions $\langle \text{gc} \rangle$ followed by an operational clause $\langle \text{oc} \rangle$. Essentially, a test list is a conjunction of simple tests $\langle \text{test} \rangle$ on the internal state, but or-branching and map searches, $\langle \text{map} \rangle$, introduce more complex conditions. The empty test list is allowed (meaning 'true'), which indicates that the alternative clause $\langle \text{ac} \rangle$ is redundant. The selective (switch) construct $\langle \text{sc} \rangle$ is also redundant, but is often more natural to use than an extended conditional.

Tests and commands (which in this text will be separated by commas in lists to aid readability) have similar syntax. The basic format is a pair $D\,v$, where D is a variable and v selects a field in the record: when this is a $\langle \text{test} \rangle$, v is the value being tested for; when this is a $\langle \text{cmd} \rangle$, v is the value assigned. Usually the fields tested are binary, in which case the modifier `x` can be used to test or assign the opposite value. Throughout, the mnemonics `l` and `f` denote the two states of a sub-route (*locked* or *free*, the modifier `x` is not used); `s` and `xs` denote the *set* or *unset* state of a route; `o`

and `c` denote track circuits *occupied* or *clear* (two separate fields in the track circuit memory). Where points are accessed one must modify the field selected with either `r` or `n` for the reverse or normal fields in points memory. The `f` modifier in P_1 `crf` is discussed later in Section 2.4.

To reduce the volume of data required the language provides a simple subroutine mechanism. In the context of a test the directive `@L` causes the interpreter to jump to the *evaluation set* identified by the label `*L` in the source; in the context of a command the label should identify an *execution set*. Evaluation sets have no side-effects, and return true or false at the point at which they occur; execution sets can be arbitrary sequences of data. Here `\` marks the end of the subroutine code, but otherwise it closes the `if` bracket. A further syntactic constraint is imposed on the use of the subroutine mechanism: the reference `@L` and the label `*L` must both appear in the same data file. (See Figure 7.3 for an example of the use of this subroutine machanism.)

The `@` directive is one of several so-called *specials*. These directives indicate to the interpreter that it should execute a pre-programmed sequence of actions, being typically given a variable name as a parameter upon which to operate. Use of the specials in preparing Geographic Data is not mandatory, but it shortens the runtime execution of the program. Further examples are given below.

Letting `*Q`, `*P`, and `*L` be metavariables over the class of labels, the *PRR*, *PFM*, and *FOP* data files can be constructed thus:

$$
\begin{array}{rcl}
\mathit{PRR} & ::= & \Big[\texttt{*Q} \Big[\langle \text{oc} \rangle \Big]^* \ \texttt{.} \quad \Big| \quad \texttt{*L} \ \langle \text{ev_set} \rangle \quad \Big| \quad \texttt{*L} \ \langle \text{ex_set} \rangle \Big]^* \\
\mathit{PFM} & ::= & \Big[\texttt{*P} \Big[\langle \text{gc} \rangle \Big]^* \ \texttt{\textbackslash} \quad \Big| \quad \texttt{*L} \ \langle \text{ev_set} \rangle \Big]^* \\
\mathit{FOP} & ::= & \Big[\langle \text{cmd} \rangle \ \langle \text{ev_set} \rangle \ \texttt{.} \quad \Big| \quad \langle \text{cs} \rangle \ \texttt{.} \Big]^*
\end{array}
$$

Each panel request and flag operation is terminated by a period. In these files the period terminates a block of data that will always be executed without interruption in the SSI minor cycle. The *PFM* data contain only tests and the label `*P` is the entry point for the interpreter when it is evaluating a points "free to move" test. The label `*Q` is the entry point for a panel request, as in `*Q4` above. Other labels are targets for jumps (`@L`) appearing in these data. There should be no jumps in the *FOP* data. (It will be convenient in the sequel to use the notation *PRR*(`*Q`) to refer to the data for the panel request `Q` $\in \mathcal{Q}$; similarly, when $P \in \mathcal{P}$ we will refer to the two data sets by *PFM*(`*PN`) and *PFM*(`*PR`), and so on.)

Our final concern in this section is with the map search data. These data, like the *PFM* data, may be accessed throughout the other source files—but typically in the *OPT* data when a search is needed to determine the proximity of a train to a signal. A *map search* is a geographic condition having the syntax specification:

$$
\langle \text{map} \rangle \quad ::= \quad \texttt{\{L} \Big[\texttt{\}L} \Big]^+
$$

The entry point for the search is the label referenced by the first entry in this list, `{L`, while the search end-points are specified by the labels referenced by the remaining elements, `}L`. The map search must specify at least one end-point. Map data are constructed thus:

$$
\begin{aligned}
\textit{MAP} &::= \left[\langle\text{segment}\rangle\right]^* \\
\langle\text{segment}\rangle &::= \texttt{*L}\ \langle\text{ref}\rangle\ \left[\langle\text{entry}\rangle\right]^*\ \langle\text{exit}\rangle \\
\langle\text{entry}\rangle &::= \langle\text{ref}\rangle \ \Big|\ \texttt{if}\ \left[\langle\text{gc}\rangle\right]^*\ \texttt{then}\ \langle\text{exit}\rangle\ \texttt{\textbackslash} \\
\langle\text{ref}\rangle &::= \#T \ \Big|\ \#S \\
\langle\text{exit}\rangle &::= \texttt{pass} \ \Big|\ \texttt{fail} \ \Big|\ \texttt{\^{}L}
\end{aligned}
$$

S and T are variables representing signals and track circuits respectively, where the specials $\#T$ and $\#S$ abbreviate simple tests on track circuit and signal memory.

Informally, a map search begins at a feature reference, usually a signal, and proceeds back through the network until a `pass` or a `fail` is encountered, or a label remembered from the beginning of the search. The conditional used in the *MAP* data is an expression, not a command as it is elsewhere. For a concrete example, the data for the search back from S_3 might include the fragment:

*T3DN $\#T_3$, $\#T_4$
    if P_2 cdr then pass \
    if P_2 cdn then ^T6DN \
    fail
*T3UP ...
*T6DN $\#T_6$, $\#T_7$
    if P_3 cdr then pass \
    if P_3 cdn then fail \
    fail

When points are encountered in the search there is a choice to be made which is governed by their current state. If these are *trailing* points in the direction of the search (*e.g.*, P_3), and if they are *controlled and detected reverse*, the search succeeds unconditionally because there can be no train approaching thew signal (S_3). If the points are *controlled and detected normal* the search continues from the location `*T6DN` in the map; otherwise the search fails (probably because the points were moving when the search started since the controlled and detected fields in points memory will normally be in correspondence). When points are *facing* the direction of the search a different principle applies, deflecting the search along one or other of the paths to the signal. The meaning of the `^L` special is discussed below.

2.4 Semantics: The Control Interpreter

The Geographic Data Language has not hitherto benefited from a formal semantics. Therefore in analysing safety properties of the data one must offer formal semantics that are faithful to the informal description of the language given in the Data Preparation Guide [9]. The informal description is inevitably vague and imprecise in places—particularly in explaining the logic encoded in the interpreter itself. It is noted, for example, that the logic encoded in the specials can be expressed in the conditional language alone, but no translation table is given to clarify the point. That this translation is not trivial is illustrated below where we discuss the relationship between the control interpreter and the points "free to move" data and the map search data in Sections 2.4.2 and 2.4.3 respectively.

2.4.1 Abstract Syntax of Simple Tests and Commands

Earlier we characterised the state of the SSI in terms of dynamic and static components. The dynamic component is what one usually means when referring to the *state* of a program, and we shall model states with the function space

$$\textbf{State} : \textbf{Var} \longrightarrow \textbf{Val}$$

i.e., mappings from a domain of variables (*e.g.*, $P, T \ldots$) to a suitable domain of values ($\{0, 1\}$ say). Let $\sigma \in$ **State**, $D \in$ **Var** be representative elements from these domains. It will also be necessary to have direct access to the static data. For this we are given a domain of labels. Let $\mathtt{L} \in$ **Lab**. Labels and references to them will be syntactically distinguished as before.

The phrase structure of the Geographic Data in the *PRR*, *PFM*, and *FOP* file (and the *MAP* file, but this is postponed until Section 2.4.3) is succinctly specified by the abstract syntax below. Let t be a test, and c a command:

$$\begin{array}{rcl} \textbf{Tst} & ::= & B \mid t_1 \,,\, t_2 \mid \mathtt{(}\, t_1 \,\mathtt{or}\, t_2 \,\mathtt{)} \mid \mathtt{@L} \\ \textbf{Cmd} & ::= & A \mid c_1 \,,\, c_2 \mid \mathtt{skip} \mid \mathtt{if}\ t\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \mid \mathtt{@L} \end{array}$$

B is a basic variable test (*e.g.*, $P_1\,\mathtt{cr}$), A is the basic command to set the value of (a field in) a variable. In order to be specific about the meaning of simple tests and commands we define interpretation functions **T** and **C**:

$$\mathbf{C} : \textbf{Cmd} \longrightarrow \textbf{State} \longrightarrow \textbf{State}$$
$$\mathbf{T} : \textbf{Tst} \longrightarrow \textbf{State} \longrightarrow (\textbf{State} \times \textbf{State}) \longrightarrow \textbf{State}$$

We expect a command to yield a new state given an initial state σ. For example, if the command is to control the points P reverse:

$$\mathbf{C}[\![P\,\mathtt{cr}]\!]\sigma \;=\; \sigma[P.\mathtt{cr} := 1] \qquad (1)$$

This (function updating) notation is chosen to emphasise that the command assigns a value to a (binary) field in the points record. Tests, on the other hand, are best understood in terms of 'continuations':

$$\mathbf{T}[\![P\ \texttt{cn}]\!]\sigma\,(s,f) \;=\; \text{if } \sigma(P.\texttt{cn}) = 1 \text{ then } s \text{ else } f \tag{2}$$

$$\mathbf{T}[\![t_1\ ,\ t_2]\!]\sigma\,(s,f) \;=\; \mathbf{T}[\![t_1]\!]\sigma\,(\mathbf{T}[\![t_2]\!]\sigma\,(s,f),f) \tag{3}$$

$$\mathbf{T}[\![\texttt{(}\ t_1\ \texttt{or}\ t_2\ \texttt{)}]\!]\sigma\,(s,f) \;=\; \mathbf{T}[\![t_1]\!]\sigma\,(s,\mathbf{T}[\![t_2]\!]\sigma\,(s,f)) \tag{4}$$

Here s is the successful continuation, and f is the failure continuation—*i.e.*, the state reached if the test fails. Notice that a test list (3) is treated as conjunction. Elaborating on this theme we obtain for the one-armed conditional:

$$\mathbf{C}[\![\texttt{if}\ t\ \texttt{then}\ c]\!]\sigma \;=\; \mathbf{T}[\![t]\!]\sigma\,(\mathbf{C}[\![c]\!]\sigma,\sigma) \tag{5}$$

If the test t is passed in state σ the commands c are executed in that state, otherwise the state remains unchanged. Command continuations may be needed to elegantly model jumps, but they do not enhance the clarity of the presentation of the language here.

2.4.2 Points Free to Move Conditions

Whenever a route is to be set over points they must first be called into the correct alignment. In SSI this is achieved in two stages: firstly, the control field in points memory must be properly set (as in the conclusion of `*Q4` above); secondly, the output telegram for the points module must be set up with the correct command (usually achieved by copying the control bits to the output). There is a problem, however, with the naïve semantics of the *points command* suggested above: $\sigma[P.\texttt{cr} := 1]$ is not the correct interpretation. According to the informal presentation of the language the control interpreter is programmed to clear the reverse control bit when the normal bit is set, and vice versa. Thus: $\mathbf{C}[\![P\ \texttt{cr}]\!]\sigma = \sigma[P.\texttt{cr} := 1][P.\texttt{cn} := 0]$, for example. Track circuit clear and occupied fields also have this inversion property, but other commands (assignments) in the language are treated uniformly as suggested by Equation (1).

The points test $P\ \texttt{crf}$ also introduces behaviour that depends on the interpreter. The test is disjunctive: it is passed if the points are already controlled reverse, and if not, it is passed if the "free to move" conditions are met. Intuitively:

$$\begin{aligned}\mathbf{T}[\![P\ \texttt{crf}]\!]\sigma\,(s,f) \;&=\; \mathbf{T}[\![\texttt{(}\ P\ \texttt{cr or}\ \textit{PFM}(\texttt{*PR})\ \texttt{)}]\!]\sigma\,(s,f)\\ &=\; \mathbf{T}[\![P\ \texttt{cr}]\!]\sigma\,(s,\mathbf{T}[\![\textit{PFM}(\texttt{*PR})]\!]\sigma\,(s,f))\end{aligned}$$

However, it should be noted that the control interpreter performs two further tests when the *PFM* data are accessed: firstly the key switch field in points memory in the opposite direction is examined; secondly, the program checks that the points have not been

disabled in the opposite direction by an override from the technician's console. The former condition can be programmed in the Geographic Data, but the latter cannot since the override flag in points memory is accessible only to the generic program. Since the override flag always *restricts* the behaviour of the SSI we shall generally ignore its effect in the sequel.

Specialising for the moment, and bringing the key switch test into consideration, we expect the following equivalence to hold:

$$P_2\,\texttt{crf} \;\equiv\; (\,P_2\,\texttt{cr or}\;P_2\,\texttt{xkn}\,,T_4^{ab}\,\texttt{f}\,,T_4^{ba}\,\texttt{f}\,,T_4\,\texttt{c}\,)$$

Similarly for the other direction of the points. More generally we obtain as the *meaning* of the points "free to move" geographic conditions:

$$\mathbf{T}[\![P\,\texttt{crf}]\!]\sigma\,(s,f) \;=\; \mathbf{T}[\![P\,\texttt{cr}]\!]\sigma\,(s,\mathbf{T}[\![P\,\texttt{xkn}\,,\mathit{PFM}(\texttt{*PR})]\!]\sigma\,(s,f)) \qquad (6)$$

$$\mathbf{T}[\![P\,\texttt{cnf}]\!]\sigma\,(s,f) \;=\; \mathbf{T}[\![P\,\texttt{cn}]\!]\sigma\,(s,\mathbf{T}[\![P\,\texttt{xkr}\,,\mathit{PFM}(\texttt{*PN})]\!]\sigma\,(s,f)) \qquad (7)$$

The substitution $\mathit{PFM}(\texttt{*PN})$ used above has been informally presented, but can be rigorously defended since the *PFM* data file is just such a function:

$$\mathit{PFM}\,:\,\mathbf{Lab}\longrightarrow\mathbf{Tst}$$

We shall generalise this notion to *MAP* and *PRR* in the sequel.

2.4.3 The Map Search

Another point of contact between the Geographic Data Language and the interpreter arises in the *MAP* data. Given a (concrete) specification of the form $\{\mathrm{L}\,,\}\mathrm{L}_1\,,\ldots\}\mathrm{L}_n$, which we shall henceforth represent by (L,E), the interpreter begins the search at location $\texttt{*}\mathrm{L}$ and terminates at one of the end-points given by the set $E=\{\mathrm{L}_1\;\ldots\,,\mathrm{L}_n\}$. More formally, let **Map** and **Ent** be new phrase classes:

$$\begin{array}{rcl}
\mathbf{Tst} & ::= & \cdots \;\mid\; (\mathrm{L},E) \\
\mathbf{Map} & ::= & \texttt{*}\mathrm{L}\;\;\#D\,,\,m \\
\mathbf{Ent} & ::= & \#D\,,\,m \;\mid\; \texttt{if}\;t\;\texttt{then}\;e\,,\,m \;\mid\; e
\end{array}$$

where m is a map entry, $D\in\mathcal{T}\cup\mathcal{S}$, and e is one of the map exits `pass`, `fail`, or $\texttt{\^{}}\mathrm{L}$. The interpretation function **M** takes a set of labels in its first argument. However, since for any map search the set of end-points is fixed, we shall write $\mathbf{M}_E[\![\cdot]\!]$ instead

$$\mathbf{M}_E\,:(\mathbf{Map}+\mathbf{Ent})\longrightarrow\mathbf{State}\longrightarrow(\mathbf{State}\times\mathbf{State})\longrightarrow\mathbf{State}$$

and define this function inductively along these lines:

$$\mathbf{T}[\![(\mathtt{L}, E)]\!]\sigma\,(s, f) = \mathbf{M}_E[\![\star\mathtt{L}\ \ \mathit{MAP}(\star\mathtt{L})]\!]\sigma\,(s, f) \quad (8)$$

$$\mathbf{M}_E[\![\star\mathtt{L}\ \ \#T\,,\ m]\!]\sigma\,(s, f) = \begin{cases} \mathbf{T}[\![T\ \mathtt{c}]\!]\sigma\,(s, f) & \text{if } \mathtt{L} \in E \\ \mathbf{M}_E[\![\#T\,,\ m]\!]\sigma\,(s, f) & \text{otherwise} \end{cases} \quad (9)$$

$$\mathbf{M}_E[\![\#T\,,\ m]\!]\sigma\,(s, f) = \mathbf{T}[\![T\ \mathtt{c}]\!]\sigma\,(\mathbf{M}_E[\![m]\!]\sigma\,(s, f), f) \quad (10)$$

$$\mathbf{M}_E[\![\mathtt{if}\ t\ \mathtt{then}\ e\,,\ m]\!]\sigma\,(s, f) = \mathbf{T}[\![t]\!]\sigma\,(\mathbf{M}_E[\![e]\!]\sigma\,(s, f), \mathbf{M}_E[\![m]\!]\sigma\,(s, f)) \quad (11)$$

From (9) note that the map search terminates if the label at the head of the list is a designated end-point (successfully or not, depending on whether the track circuit referenced is clear); otherwise the search continues along the map m if the track circuit is clear, or fails if it is not by (10). A signal reference $\#S$ is treated in a similar manner, being also an abbreviation for a simple test on the signal memory. Clause (11) is what one would expect for an *if-then-else* expression. The end-point rules are simpler:

$$\mathbf{M}_E[\![\mathtt{pass}]\!]\sigma\,(s, f) = s \quad (12)$$

$$\mathbf{M}_E[\![\mathtt{fail}]\!]\sigma\,(s, f) = f \quad (13)$$

$$\mathbf{M}_E[\![\hat{}\mathtt{L}]\!]\sigma\,(s, f) = \mathbf{M}_E[\![\star\mathtt{L}\ \ \mathit{MAP}(\star\mathtt{L})]\!]\sigma\,(s, f) \quad (14)$$

Thus, when the special $\hat{}\mathtt{L}$ is encountered the interpreter jumps to the indicated label in the map—of course, this means that there is no guarantee that a map search ever terminates since one can easily define a cyclic map segment. In order to ensure that the interpretation functions are total we suppose that the data are well formed in the sense that cyclic references are syntactically prohibited (which is the case, in fact). Note that only one branch in the map is explored for any search conducted: no backtracking is necessary since there can be at most one open path to a signal at any time.

2.5 Indirect Semantics of the Map Search

The interpretation function **M** defines the algorithm that the control interpreter should perform when encountering a map search $(\mathtt{L}, E)$ geographic condition—that is, **M** specifies how to conduct the search dynamically. When we come, as in Chapter 5, to *formalise* the syntax and semantics of the language, this direct interpretation of the map search will be inconvenient to manipulate because of the need to represent the map in the formalism (as well as the algorithm). However, since the set E is defined statically, we can convert the map into a decision tree *a priori* and reason with that instead. This supplies an *indirect* semantics for the map search, which this section justifies.

To give a rigorous context we extend the phrase class **Tst** with pseudo tests for the *if-then-else* form, and the constants `pass` and `fail`. Then we extend the definition of

the semantic valuation function **T** by defining

$$\mathbf{T}[\![\texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3]\!]\sigma\,(s,f) = \mathbf{T}[\![t_1]\!]\sigma\,(\mathbf{T}[\![t_2]\!]\sigma\,(s,f), \mathbf{T}[\![t_3]\!]\sigma\,(s,f)) \quad (15)$$

$$\mathbf{T}[\![\texttt{pass}]\!]\sigma\,(s,f) = s \quad (16)$$

$$\mathbf{T}[\![\texttt{fail}]\!]\sigma\,(s,f) = f \quad (17)$$

and syntactically translate $t \in$ **Tst** which may have a map search, to $\{\!|t|\!\} \in$ **Tst** which has the map search converted to *if-then-else* normal form. The function $\{\!|\cdot|\!\}$ is the identity everywhere, except:

$$\{\!|(\mathrm{L}, E)|\!\} = \{\!|\texttt{*}\mathrm{L}\ \ \mathit{MAP}(\texttt{*}\mathrm{L})|\!\}_E \quad \text{(i)}$$

$$\{\!|\texttt{*}\mathrm{L}\ \ \#D\,,\, m|\!\}_E = \begin{cases} \#D & \text{if } \mathrm{L} \in E \\ \{\!|\#D\,,\, m|\!\}_E & \text{otherwise} \end{cases} \quad \text{(ii)}$$

$$\{\!|\#D\,,\, m|\!\}_E = \texttt{if}\ \#D\ \texttt{then}\ \{\!|m|\!\}_E\ \texttt{else}\ \texttt{fail} \quad \text{(iii)}$$

$$\{\!|\texttt{pass}|\!\}_E = \texttt{pass} \quad \text{(iv)}$$

$$\{\!|\texttt{fail}|\!\}_E = \texttt{fail} \quad \text{(v)}$$

$$\{\!|\texttt{\^{}}\mathrm{L}|\!\}_E = \{\!|\texttt{*}\mathrm{L}\,,\, \mathit{MAP}(\texttt{*}\mathrm{L})|\!\}_E \quad \text{(vi)}$$

$$\{\!|\texttt{if}\ t\ \texttt{then}\ e\,,\, m|\!\}_E = \texttt{if}\ \{\!|t|\!\}\ \texttt{then}\ \{\!|e|\!\}_E\ \texttt{else}\ \{\!|m|\!\}_E \quad \text{(vii)}$$

In the last clause defining $\{\!|\cdot|\!\}_E$ we apply the (unadorned) transformation $\{\!|\cdot|\!\}$ to the guard t (*cf.* Equation (11)). Since the *MAP* data are required to be free of (syntactic) cyclic references $\{\!|t|\!\}$ is always defined (and finite).

Theorem 2.1 The direct and the indirect semantics of the map search agree:

$$\mathbf{T}[\![\{\!|(\mathrm{L}, E)|\!\}]\!]\sigma\,(s,f) = \mathbf{T}[\![(\mathrm{L}, E)]\!]\sigma\,(s,f)$$

when $\{\!|(\mathrm{L}, E)|\!\}$ is defined, for any σ, s, and f. □

Proof Theorem 2.1 is a corollary to $\mathbf{T}[\![t]\!] = \mathbf{T}[\![\{\!|t|\!\}]\!]$, which is proved by induction of the structure of tests t, and the depth of the decision tree. The interesting case is for the map search where one has to show

$$\mathbf{T}[\![\{\!|(\mathrm{L}, E)|\!\}]\!]\sigma\,(s,f) = \mathbf{T}[\![(\mathrm{L}, E)]\!]\sigma\,(s,f)$$

$$\mathbf{T}[\![\{\!|\texttt{*}\mathrm{L}\ \ \mathit{MAP}(\texttt{*}\mathrm{L})|\!\}_E]\!]\sigma\,(s,f) = \mathbf{M}_E[\![\texttt{*}\mathrm{L}\ \ \mathit{MAP}(\texttt{*}\mathrm{L})]\!]\sigma\,(s,f)$$

(applying the rules given above and in Section 2.4.3) which proceeds by induction on the structure of maps.

base cases These are trivial, by the definitions (12,16,iv) and (13,17,v):

$$\mathbf{M}_E[\![\texttt{pass}]\!]\sigma\,(s,f) = s = \mathbf{T}[\![\{\!|\texttt{pass}|\!\}_E]\!]\sigma\,(s,f)$$

$$\mathbf{M}_E[\![\texttt{fail}]\!]\sigma\,(s,f) = f = \mathbf{T}[\![\{\!|\texttt{fail}|\!\}_E]\!]\sigma\,(s,f)$$

case $\texttt{*L}\ \ \texttt{\#}D\,,\ m$. There are two subcases to consider. When $\texttt{L} \in E$ both sides reduce to $\mathbf{T}[\![\texttt{\#}D]\!]\sigma\,(s,f)$ by (9) and (ii). Otherwise $\texttt{L} \notin E$, and the induction hypothesis will be $\mathbf{T}[\![\{\!|m|\!\}_E]\!]\sigma\,(s,f) = \mathbf{M}_E[\![m]\!]\sigma\,(s,f)$. Then

$$\mathbf{M}_E[\![\texttt{*L}\ \ \texttt{\#}D\,,\ m]\!]\sigma\,(s,f) \quad = \quad \mathbf{T}[\![\texttt{\#}D]\!]\sigma\,(\mathbf{M}_E[\![m]\!]\sigma\,(s,f),f)$$

by (9), whereas the expanded term yields by way of (15), (ii) and (iii):

$$\begin{aligned}\mathbf{T}[\![\{\!|\texttt{*L}\ \ \texttt{\#}D\,,\ m|\!\}_E]\!]\sigma\,(s,f) &= \mathbf{T}[\![\{\!|\texttt{\#}D\,,\ m|\!\}_E]\!]\sigma\,(s,f)\\ &= \mathbf{T}[\![\texttt{if \#}D\ \texttt{then}\ \{\!|m|\!\}_E\ \texttt{else fail}]\!]\sigma\,(s,f)\\ &= \mathbf{T}[\![\texttt{\#}D]\!]\sigma\,(\mathbf{T}[\![\{\!|m|\!\}_E]\!]\sigma\,(s,f),f)\end{aligned}$$

Applying the induction hypothesis proves the result in this case.

case $\texttt{\#}D\,,\ m$. This is similar to the above when $\texttt{L} \notin E$.

case $\texttt{\^{}L}$. The result follows by finiteness of the map and the assumption that $\{\!|t|\!\}$ is defined for any t (hence so is $\{\!|m|\!\}_E$, for any map m). Consequently

$$\mathbf{M}_E[\![\texttt{*L}\ \ \mathit{MAP}(\texttt{*L})]\!]\sigma\,(s,f) \quad = \quad \mathbf{T}[\![\{\!|\texttt{*L}\ \ \mathit{MAP}(\texttt{*L})|\!\}_E]\!]\sigma\,(s,f)$$

by a shorter inference, so $\mathbf{M}_E[\![\texttt{\^{}L}]\!]\sigma\,(s,f) = \mathbf{T}[\![\{\!|\texttt{\^{}L}|\!\}_E]\!]\sigma\,(s,f)$.

case $\texttt{if}\ t\ \texttt{then}\ e\,,\ m$. Two induction hypotheses are needed: $\mathbf{T}[\![\{\!|m|\!\}_E]\!]\sigma\,(s,f) = \mathbf{M}_E[\![m]\!]\sigma\,(s,f)$, and $\mathbf{T}[\![\{\!|e|\!\}_E]\!]\sigma\,(s,f) = \mathbf{M}_E[\![e]\!]\sigma\,(s,f)$. Then

$$\begin{aligned}&\mathbf{M}_E[\![\texttt{if}\ t\ \texttt{then}\ e\,,\ m]\!]\sigma\,(s,f)\\ &\quad = \quad \mathbf{T}[\![t]\!]\sigma\,(\mathbf{M}_E[\![e]\!]\sigma\,(s,f),\mathbf{M}_E[\![m]\!]\sigma\,(s,f))\end{aligned}$$

by (11), while

$$\begin{aligned}&\mathbf{T}[\![\{\!|\texttt{if}\ t\ \texttt{then}\ e\,,\ m|\!\}_E]\!]\sigma\,(s,f)\\ &\quad = \quad \mathbf{T}[\![\{\!|t|\!\}]\!]\sigma\,(\mathbf{T}[\![\{\!|e|\!\}_E]\!]\sigma\,(s,f),\mathbf{T}[\![\{\!|m|\!\}_E]\!]\sigma\,(s,f))\end{aligned}$$

using (15) and (vii). Applying the induction hypothesis, and generalising, leaves the requirement to prove that

$$\mathbf{T}[\![t]\!]\sigma\,(s',f') \quad = \quad \mathbf{T}[\![\{\!|t|\!\}]\!]\sigma\,(s',f')$$

This follows by a shorter inference by finiteness of $\{\!|t|\!\}$.

Hence $\mathbf{T}[\![t]\!] = \mathbf{T}[\![\{\!|t|\!\}]\!]$, and Theorem 2.1 follows. ■

The assumption that $\{\!|\cdot|\!\}$ is a total function seems quite strong, but this is really a matter of pragmatics: the data compiler has to check for the possible existence of cyclic references in the *MAP* data (also in the other data source files) when the data are loaded into the SSI. The syntactic check is the most practical means of doing this,

$$
\begin{array}{lcl}
\mathbf{C}[\![D\ v]\!]\sigma & = & \sigma[D.v := 1] \\
\mathbf{C}[\![D\ \mathtt{x}v]\!]\sigma & = & \sigma[D.v := 0] \\
\mathbf{C}[\![P\ \mathtt{cr}]\!]\sigma & = & \sigma[P.\mathtt{cr} := 1][P.\mathtt{cn} := 0] \\
\mathbf{C}[\![P\ \mathtt{cn}]\!]\sigma & = & \sigma[P.\mathtt{cn} := 1][P.\mathtt{cr} := 0] \\
\mathbf{C}[\![\mathtt{skip}]\!]\sigma & = & \sigma \\
\mathbf{C}[\![c_1\ ,\ c_2]\!]\sigma & = & \mathbf{C}[\![c_2]\!](\mathbf{C}[\![c_1]\!]\sigma) \\
\mathbf{C}[\![\mathtt{@L}]\!]\sigma & = & \mathbf{C}[\![\mathit{PRR}(\mathtt{*L})]\!]\sigma \\
\mathbf{C}[\![\mathtt{if}\ t\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2]\!]\sigma & = & \mathbf{T}[\![t]\!]\sigma\ (\mathbf{C}[\![c_1]\!]\sigma, \mathbf{C}[\![c_2]\!]\sigma) \\
 & & \\
\mathbf{T}[\![D\ v]\!]\sigma\ (s,f) & = & \text{if } \sigma(D.v) = 1 \text{ then } s \text{ else } f \\
\mathbf{T}[\![D\ \mathtt{x}v]\!]\sigma\ (s,f) & = & \text{if } \sigma(D.v) = 0 \text{ then } s \text{ else } f \\
\mathbf{T}[\![P\ \mathtt{cn}]\!]\sigma\ (s,f) & = & \text{if } \sigma(P.\mathtt{cn}) = 1 \text{ then } s \text{ else } f \\
\mathbf{T}[\![P\ \mathtt{kn}]\!]\sigma\ (s,f) & = & \text{if } \sigma(P.\mathtt{kn}) = 1 \text{ then } s \text{ else } f \\
\mathbf{T}[\![P\ \mathtt{crf}]\!]\sigma\ (s,f) & = & \mathbf{T}[\![P\ \mathtt{cr}]\!]\sigma\ (s, \mathbf{T}[\![P\ \mathtt{xkn}\ , \mathit{PFM}(\mathtt{*PR})]\!]\sigma\ (s,f)) \\
\mathbf{T}[\![P\ \mathtt{cnf}]\!]\sigma\ (s,f) & = & \mathbf{T}[\![P\ \mathtt{cn}]\!]\sigma\ (s, \mathbf{T}[\![P\ \mathtt{xkr}\ , \mathit{PFM}(\mathtt{*PN})]\!]\sigma\ (s,f)) \\
\mathbf{T}[\![t_1\ ,\ t_2]\!]\sigma\ (s,f) & = & \mathbf{T}[\![t_1]\!]\sigma\ (\mathbf{T}[\![t_2]\!]\sigma\ (s,f), f) \\
\mathbf{T}[\![\mathtt{(}\ t_1\ \mathtt{or}\ t_2\ \mathtt{)}]\!]\sigma\ (s,f) & = & \mathbf{T}[\![t_1]\!]\sigma\ (s, \mathbf{T}[\![t_2]\!]\sigma\ (s,f)) \\
\mathbf{T}[\![\mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3]\!]\sigma\ (s,f) & = & \mathbf{T}[\![t_1]\!]\sigma\ (\mathbf{T}[\![t_2]\!]\sigma\ (s,f), \mathbf{T}[\![t_3]\!]\sigma\ (s,f)) \\
\mathbf{T}[\![\mathtt{pass}]\!]\sigma\ (s,f) & = & s \\
\mathbf{T}[\![\mathtt{fail}]\!]\sigma\ (s,f) & = & f
\end{array}
$$

Figure 2.3: Semantics of the conditional language

although eliminating potentially interesting 'maps' that contain syntactic cycles, but not semantic ones. The question of whether semantic cycles exist is a difficult issue that is related to the problem of eliminating *causal loops* (short-circuits) in sequential hardware. Malik [55] has a polynomial algorithm to decide whether a sequential circuit can be converted into a (much larger) *combinational* circuit: this algorithm may be adapted to the present setting by representing the map (graph) as a system of Boolean equations, but the syntactic test for circularity is presently acceptable in practice.

2.6 Summary

This brings to a close our examination of the syntax and semantics of the conditional language in which Geographic Data are specified. Henceforth we shall assume that map searches (and *evaluation sets* which can be treated similarly) have been eliminated from the data analysed in the manner suggested in the preceding section. The main semantic definitions are summarised in Figure 2.3. Note that $c_1\ ,\ c_2$ represents sequential composition, and that the x flag in (basic) tests and commands is interpreted

as inverting the literal value: negation can only be applied at this level. Note, too, that expressions are only of Boolean type. The Geographic Data has access to other data, such as counters (*elapsed timers*), but no means of calculating with these other than by comparison with integral constants (or enumerated type constants in the case of signal aspects, say).

Not all features of the Geographic Data Language have been summarised above. In particular, the *OPT* data have been omitted since they do not play a rôle in later chapters. However, these data are also built from conditional and sequential constructs. We note that there is no looping construct, since none is needed to specify the interlocking logic. Implicit loops can be defined in terms of jumps (the `@L` special) to execution sets, but this practice is forbidden by the data compiler. Recall that the Geographic Data are evaluated in real-time, subject to stringent timing constraints, and loops may have unpredictable timing behaviour (or at least timing properties that are difficult to verify).

Even so, interlocking behaviour may be very complex. In the next chapter a CCS model of Solid State Interlocking is developed which provides a framework in which it is intended to prove safety properties of the data. The model is derived by a translation according to the above semantics. We focus on *properties* of the data, and on the problem of proving these properties for simplified signalling systems such as WEST. These early attempts to (mechanically) verify properties of Geographic Data are refined in later chapters to a stage where we can verify properties of real signalling data.

Chapter 3

Modelling Solid State Interlocking

This chapter builds on the informal description of Solid State Interlocking given in the Introduction. The model developed in Section 3.2 serves as a reference for this and subsequent chapters so some time is taken below to discuss its objectives and particular representation. The model is derived from a translation of the Geographic Data into CCS. Ultimately this gives rise to a *labelled transition system*, an automaton whose safety properties are formally characterised in Section 3.3 as *invariants* expressed in the modal μ-calculus. Then in Sections 3.4 and 3.5 we address the question of verifying that these invariants hold for some simple examples. The problem of finding a flexible and efficient framework within which to conduct the formal verification, which is scalable and which will also form the foundation of a *mechanical* Geographic Data checker, will occupy us through to Chapter 5.

3.1 Introduction

The purpose of railway signalling is to ensure the safe passage of trains through topologically complex networks. Safety properties therefore dominate our analysis. In designing and constructing signalling systems engineers seek to ensure, amongst other things, the following safety properties of the system *as a whole*:

- At no time does more than one train occupy a given track section (except when a train is being coupled to an engine of course).
- No train passes through a set of points which are not locked (locking the points physically prevents their movement).
- No train passes in the normal (or reverse) direction through trailing points which are locked reverse (or normal).

These have been drawn from discussions with signalling engineers. To understand the last of these, consider the fragment from WEST in Figure 3.1. If a train is moving

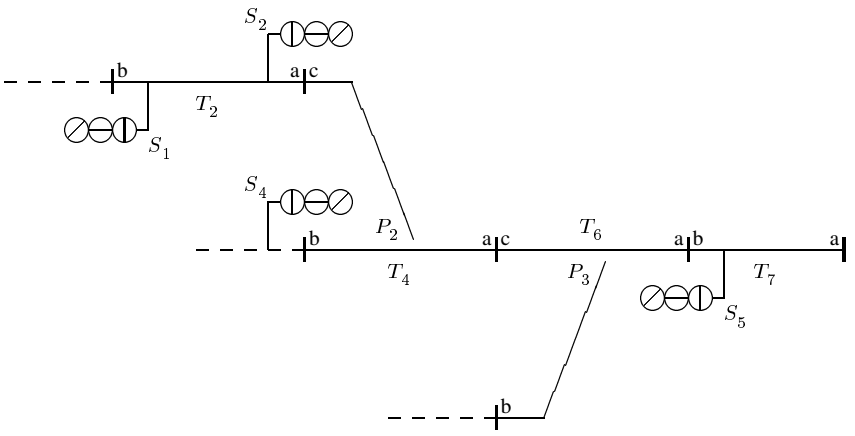


Figure 3.1: Trailing points may derail trains if locked against the direction of travel

through points P_2 in the direction $\vec{ba}$ (normal) there is a real danger of its derailment if the points are in fact (physically) clamped in the reverse ($\vec{ca}$) orientation; derailment is also likely if the points are moved while a train is passing through T_4.

The problem, of course, is that properties such as these plainly rely on the behaviour of trains, or at least on their drivers, and are therefore very unlikely to be demonstrable in practice because trains may fail to obey signals. But the engineering problem is to ensure that *in principle* no two trains are allowed simultaneous access to the same section of track, and so on. This immediately focuses attention on the interlocking logic, and obviates the need to capture in a formal model vagaries in the behaviour of trains, elements of the network, or the complex communications between the signal control centre and the railway. We do not attempt to verify that *at no time does more than one train occupy a given track section* because any model in which it makes sense to express such a property at all would be exceedingly difficult (if not impossible) to validate with respect to the physical control system.

Nevertheless, the correct functioning of the underlying communication mechanism is intrinsic to the safe operation of SSI as a whole. There is considerable scope for formally verifying safety properties of the SSI generic software (see Cribbens [24], and Chapter 6 for example), although a thorough analysis of the underlying communication mechanisms is an issue whose scope is too broad for a completely formal understanding. Yet even if the communications between SSI and track-side equipment are functioning perfectly, it is clear that the presence of errors in the Geographic Data will negate the Interlocking's overall integrity.

It is therefore argued here that the most fundamental safety properties of the signal control system may be exposed in the Geographic Data alone. An example from Section 3.3 is the *mutual exclusion* property: no more than one of the sub-routes over a given track section is locked at any time. This says something about the logical rela-

tionship between the control variables that constitute the Interlocking's internal representation of the state of the railway: it expresses an *invariant* of the program's memory. Admittedly this property does not hold while the SSI is in its initialisation phase, but if it holds thereafter it lends considerable credence to the notion that *in principle* no two trains have simultaneous access to a given section of track. In the final analysis however, it is difficult to argue (formally) that properties such as this are sufficient, in the absence of faults elsewhere, to guarantee system properties such as those identified above. So for the present we contend only that these properties of the Geographic Data are interesting in themselves.

Properties of the data such as the mutual exclusion property above can be verified only if a suitable semantics for the Geographic Data Language is given. In Section 3.2 these semantics are given indirectly by translating the Geographic Data into CCS, following the definitions in the preceding chapter. Consequently we obtain an operational semantics, and require only to validate the operational model with respect to the physical system—in this case the control interpreter. In fact, the model derived in this way is rather abstract so it is better thought of as a *formal specification* of the admissible behaviour (state transitions) of the SSI: details such as the order in which the data are to be executed, and how and when communications are initiated, are omitted.

Clearly there are many ways in which one could present a formal model of SSI, its Geographic Data, and properties of the system. Some of the approaches that have been tried were mentioned in Section 1.5. Our choice of CCS seems strange at first since this is a language for modelling parallel systems. While there is loose parallelism in the interactions between Interlockings, there is no parallelism within a single SSI—only a sequential control program. However, it turns out that the model obtained by interpreting the Geographic Data in CCS has precisely the execution model needed to verify that they are safe. As we shall see in Chapter 4, as far as the invariance *proof* is concerned it makes little difference whether one represents the model, say, in TLA, CCS, or in UNITY. Moreover, the model developed in here turns out to be easy to extend when, in Chapter 6, the problem of verifying properties of the inter-SSI communications protocol is addressed. It is in this area, that of protocol verification, that CCS and its derivatives have been shown to be most successful.

The mechanical support for CCS is rudimentary [20], but the advantage of the Concurrency Workbench (CWB) is that the tool can be used: to simulate the model, single stepping through the execution; to test the model in the sense of Hennessey's [41] equivalences and preorders; and to verify temporal properties of the system through model checking. The versatility of the tool is due to a simple representation of the model (*i.e.*, the transition system, as a graph). The techniques discussed in Section 3.4 for controlling the model's space complexity are highly effective, but the tool's simple

representation of the model is also a weakness and, it turns out, in general too concrete for large scale applications.

Safety properties of the Geographic Data are captured by logical relationships between the control variables. Technically, these will be expressed as state formulae in the modal μ-calculus: the formulae will be true of a state if and only if the state is safe—this *defines* safety. In Section 3.3 we formalise the safety properties and formulate the proof obligation which is to show that all (reachable) states of the model are safe. Since formulae of the modal μ-calculus are interpreted over transition systems the logic is well suited to specification of CCS programs. The logic also subsumes the familiar program verification logics like Floyd-Hoare logic and PDL [11, 95]. The specification style of Floyd-Hoare logic is particularly suitable for proving safety properties of Geographic Data (see Chapter 5). We start out, therefore, with a quite general framework in which to model SSI.

3.2 CCS Model of Solid State Interlocking

A natural starting point for a CCS model is to consider the system as a whole, decomposing its overall behaviour into that of two communicating systems:

$$(\mathsf{Interlocking} \mid \mathsf{Network}) \backslash L$$

Network is a parallel composition of many agents representing physical components of the system such as signals and points. In this design we may choose to abstract features of the communications mechanism, allowing synchronisations on the actions L to represent the transfer of telegrams between SSI and track-side functional modules. However the Network component in the above scheme is redundant when our primary concern is with the properties of the Geographic Data rather than the overall behaviour of the system. Such a model is needed to animate the behaviour allowed by the control system—to test and simulate the design of the interlocking—but the focus here is on the Interlocking model and its underlying assumptions.

3.2.1 Modelling Assumptions

Conceptually, the SSI model consists of three parts: the *interpreter*, the *data*, and the program *memory*. The latter component represents the Interlocking's internal state—the image of the railway—and consists of the collection of all control variables defined for the system. The former components, which are referred to collectively in the sequel as *the control*, embody not only the Geographic Data but also the assumptions we make about the behaviour of the interpreter: we would prefer these assumptions to be as weak as possible.

⟨pfm⟩	::=	⟨tl⟩
⟨fop⟩	::=	⟨cmd⟩ `if` ⟨tl⟩
⟨prr⟩	::=	`if` ⟨tl⟩ `then` ⟨cl⟩
⟨tl⟩	::=	⟨test⟩ [⟨tl⟩]
⟨test⟩	::=	⟨gc⟩ | (⟨tl⟩ `or` ⟨tl⟩)
⟨cl⟩	::=	⟨cmd⟩ [⟨cl⟩]
⟨gc⟩	::=	P (`cn[f]` | `cr[f]`) | R (`s` | `xs`) | T (`o` | `c`) | U (`l` | `f`)
⟨cmd⟩	::=	P (`cn` | `cr`) | R (`s` | `xs`) | T (`o` | `c`) | U (`l` | `f`)

Figure 3.2: Simple grammar for a subset of the Geographic Data Language

In Section 2.4 the relationship between the SSI generic program and the Geographic Data was discussed in some detail, at least in respect of *PFM* and *MAP* data. In developing the model below we shall assume the validity of those semantics, particularly the points "free to move" test. Otherwise, the principal assumption in our analysis is that the safety properties of the data do not depend on the generic program.

It is important to note that the signal engineer has complete authority in specifying the execution order imposed on the Geographic Data—*i.e.*, in defining the polling cycle which is not itself expressible in the Geographic Data Language. We contend therefore that the safety properties of the data are, or at least ought to be, independent of the order imposed. As a corollary, properties that can be proved under the assumption of an arbitrary execution order will be enjoyed by any system implementing those data—because to assume an arbitrary execution order is to assume *nothing* about the order. Under these conditions one has only to perform the safety analysis once: otherwise one would be obliged to redo the proof whenever the execution order is changed, or a rule in the database is modified.

Other assumptions require less comment. Firstly, we abstract the system's outputs to, and inputs from, the railway, and indeed follow the informal specification [66] closely in ignoring the *IPT* and *OPT* data and concentrating on the route locking data in the *PRR*, *PFM* and *FOP* data files. Secondly, the image of the railway is described by a collection of *Boolean* variables representing points, track circuits, routes and subroutes. For the model this simplification is inessential, but is mandated to some extent by the desire to control the computational complexity in automating the formal analysis. Finally, and in accordance with the above, it is appropriate for the time being to suppose the somewhat restricted syntax for Geographic Data given in Figure 3.2. While this guarded command fragment is inexpressive as a programming language, it is sufficient to encode many examples in the *FOP* and *PRR* data files.

3.2.2 Model

In his book [62] Milner defines the semantics of a simple (concurrent) imperative programming language by translating it into CCS. In this way a program is represented (concretely) by a labelled transition system. Milner's objective here is to interpret one calculus, that of Hoare logic and its proof system for the language in question, in his more primitive calculus [63]. The semantic embedding so derived demonstrates the universality of CCS, and is used by Milner to investigate properties of the programming language and its program logic. Here, we shall simply use this form of embedding in order to fix the (operational) semantics of the Geographic Data Language.

Although its designers may not think of the Geographic Data Language as a programming language as such, the presence of the interpreter provides a convenient execution model—so it might as well be assumed that we 'run' the data. More precisely, each rule in the database defines a state transformation, where a state is the Interlocking's current image of the railway. The execution model can be described in a single command loop—an endless **do** loop in Dijkstra's language of guarded commands, say. In CCS this becomes a single recursive agent which nondeterministically chooses among the guarded commands to execute at each iteration.

The typical scheme for representing a program variable as a readable and writable location in memory is given by the following pair of agents:

$$\begin{aligned} \mathsf{Loc}_D &\stackrel{\text{def}}{=} \mathsf{put}_D(x).\mathsf{Reg}_D(x) \\ \mathsf{Reg}_D(y) &\stackrel{\text{def}}{=} \mathsf{put}_D(x).\mathsf{Reg}_D(x) + \overline{\mathsf{get}}_D(y).\mathsf{Reg}_D(y) \end{aligned}$$

The values the program variable D may take are drawn from some (usually finite) data domain. For convenience it will be assumed in the model that these registers are always suitably initialised before being read. In what follows the CCS value passing syntax would quickly become cumbersome to use in its full formality—because of a profusion of sub- and super-scripts. We shall therefore abuse the notation slightly, and represent the binary track circuit variable T, for example, by the pair agents:

$$\begin{aligned} \mathsf{Reg}_T(\mathtt{c}) &\stackrel{\text{def}}{=} \mathsf{put}_T(\mathtt{c}).\mathsf{Reg}_T(\mathtt{c}) + \mathsf{put}_T(\mathtt{o}).\mathsf{Reg}_T(\mathtt{o}) + \overline{\mathsf{get}}_T(\mathtt{c}).\mathsf{Reg}_T(\mathtt{c}) \\ \mathsf{Reg}_T(\mathtt{o}) &\stackrel{\text{def}}{=} \mathsf{put}_T(\mathtt{o}).\mathsf{Reg}_T(\mathtt{o}) + \mathsf{put}_T(\mathtt{c}).\mathsf{Reg}_T(\mathtt{c}) + \overline{\mathsf{get}}_T(\mathtt{o}).\mathsf{Reg}_T(\mathtt{o}) \end{aligned}$$

The *name* $\mathsf{put}_T(\mathtt{c})$ denotes the pure CCS action obtained by the usual interpretation of the value passing calculus in the basic calculus (see Appendix B); $\overline{\mathsf{put}}_T(\mathtt{c})$ denotes the inverse action. We shall freely use $\overline{\mathsf{put}}_T(v)$, *etc.*, when the value communicated need not be specified like this, or when the value passing syntax offers greater clarity.

Using the registers defined above the image of the railway is represented by

$$\mathsf{Image} \stackrel{\text{def}}{=} \prod_{U \in \mathcal{U}} \mathsf{Reg}_U(\mathtt{f}) \mid \prod_{R \in \mathcal{R}} \mathsf{Reg}_R(\mathtt{xs}) \mid \prod_{T \in \mathcal{T}} \mathsf{Reg}_T(\mathtt{c}) \mid \prod_{P \in \mathcal{P}} \mathsf{Reg}_P(\mathtt{cn})$$

$$
\begin{array}{rcl}
\mathsf{Reg}_D(y) & \stackrel{\text{def}}{=} & \mathsf{put}_D(x).\mathsf{Reg}_D(x) + \overline{\mathsf{get}}_D(y).\mathsf{Reg}_D(y) \\
\mathsf{Image} & \stackrel{\text{def}}{=} & \prod_{U \in \mathcal{U}} \mathsf{Reg}_U(\mathtt{f}) \mid \prod_{R \in \mathcal{R}} \mathsf{Reg}_R(\mathtt{xs}) \mid \prod_{T \in \mathcal{T}} \mathsf{Reg}_T(\mathtt{c}) \mid \prod_{P \in \mathcal{P}} \mathsf{Reg}_P(\mathtt{cn}) \\
\mathsf{Control} & \stackrel{\text{def}}{=} & \mathsf{Request} + \mathsf{Unlock} + \mathsf{Cancel} + \mathsf{Input} \\
\mathsf{Request} & \stackrel{\text{def}}{=} & \sum_{Q \in \mathcal{Q}} \mathsf{set}_Q.(\mathbf{C}[\![\mathit{PRR}(\star\mathtt{Q})]\!]\mathsf{Control}) \\
\mathsf{Unlock} & \stackrel{\text{def}}{=} & \sum_{U \in \mathcal{U}} (\mathbf{C}[\![\mathit{FOP}(U)]\!]\mathsf{Control}) \\
\mathsf{Cancel} & \stackrel{\text{def}}{=} & \sum_{R \in \mathcal{R}} \mathsf{can}_R.\overline{\mathsf{put}}_R(\mathtt{xs}).\mathsf{Control} \\
\mathsf{Input} & \stackrel{\text{def}}{=} & \sum_{T \in \mathcal{T}} \mathsf{in}_T(\mathtt{o}).\overline{\mathsf{put}}_T(\mathtt{o}).\mathsf{Control} \; + \; \mathsf{in}_T(\mathtt{c}).\overline{\mathsf{put}}_T(\mathtt{c}).\mathsf{Control} \\
\mathsf{West} & \stackrel{\text{def}}{=} & (\mathsf{Control} \mid \mathsf{Image}) \backslash L
\end{array}
$$

Model #1: A CCS model of Solid State Interlocking

This is a parallel combination of agents, though none of the components communicate. The image of the railway has been initialised to some suitable state, and $\mathcal{D} = \mathcal{P} \uplus \mathcal{R} \uplus \mathcal{T} \uplus \mathcal{U}$ represents the set of WEST's points, routes, track circuits and sub-routes.

The execution model proposed is simple: during each iteration of the Interlocking's minor cycle one of the rules in the database is selected at random and evaluated. Only if the guards are currently true will the state be updated by the commands in the conclusion of the rule. This behaviour is encoded in the recursive

$$\mathsf{Control} \stackrel{\text{def}}{=} \mathsf{Request} + \mathsf{Unlock} + \mathsf{Cancel} + \mathsf{Input}$$

Each of the sequential components embodies one aspect of the behaviour of the system; each returns evaluation to the Control state which we may think of as the 'top' of the minor cycle loop in the system's ongoing evaluation of the data. The elements of this model are gathered together in the agent West displayed in Model #1 (which may sometimes be referred to explicitly as $\mathsf{West}_{\#1}$ in the sequel). The Request and Unlock components are derived by translating the Geographic Data into CCS as described below. Other components allow one to cancel routes that have been set, and to follow the movements of trains in the network. The restriction set $L = \mathcal{L}(\mathsf{Image})$ binds the components of the model together. The visible actions are:

- $\{\mathsf{set}_Q \mid Q \in \mathcal{Q}\}$, representing panel route requests. Each of these invokes the appropriate rule in the *PRR* data file, one per route. In SSI these are always processed in a single minor cycle.

- $\{\mathsf{can}_R \mid R \in \mathcal{R}\}$, representing route cancellation requests. These are also panel requests: there are normally several preconditions to be satisfied before the route can be unset, but this is not captured in the model.

$$
\begin{array}{rcll}
\mathbf{C}[\![\texttt{if}\ \langle\text{tl}\rangle\ \texttt{then}\ \langle\text{cl}\rangle\ ]\!]c & = & \mathbf{T}[\![\ \langle\text{tl}\rangle\ ]\!](\mathbf{C}[\![\ \langle\text{cl}\rangle\ ]\!]c, c) & (1)\\
\mathbf{C}[\![\ \langle\text{cmd}\rangle\ \texttt{if}\ \langle\text{tl}\rangle\ ]\!]c & = & \mathbf{T}[\![\ \langle\text{tl}\rangle\ ]\!](\mathbf{C}[\![\ \langle\text{cmd}\rangle\ ]\!]c, c) & (2)\\
\mathbf{C}[\![\ ]\!]c & = & c & (3)\\
\mathbf{C}[\![D\ v\ \langle\text{cl}\rangle\ ]\!]c & = & \overline{\mathsf{put}}_D(v).\mathbf{C}[\![\ \langle\text{cl}\rangle\ ]\!]c \qquad D \in \mathcal{D} & (4)\\
\mathbf{T}[\![\ ]\!](s, f) & = & s & (5)\\
\mathbf{T}[\![\ D\ v\ ]\!](s, f) & = & \mathsf{get}_D(u).\mathit{if}\ (u = v)\ \mathit{then}\ s\ \mathit{else}\ f \qquad D \in \mathcal{D} & (6)\\
\mathbf{T}[\![\ P\ \texttt{crf}\ ]\!](s, f) & = & \mathbf{T}[\![\ (P\ \texttt{cr or}\ \mathit{PFM}(\texttt{*PR}))\ ]\!](s, f) & (7)\\
\mathbf{T}[\![\ P\ \texttt{cnf}\ ]\!](s, f) & = & \mathbf{T}[\![\ (P\ \texttt{cn or}\ \mathit{PFM}(\texttt{*PN}))\ ]\!](s, f) & (8)\\
\mathbf{T}[\![\ \langle\text{test}\rangle\ \langle\text{tl}\rangle\ ]\!](s, f) & = & \mathbf{T}[\![\ \langle\text{test}\rangle\ ]\!](\mathbf{T}[\![\ \langle\text{tl}\rangle\ ]\!](s, f), f) & (9)\\
\mathbf{T}[\![\ (\ \langle\text{tl}\rangle_1\ \texttt{or}\ \langle\text{tl}\rangle_2\ )\ ]\!](s, f) & = & \mathbf{T}[\![\ \langle\text{tl}\rangle_1\ ]\!](s, \mathbf{T}[\![\ \langle\text{tl}\rangle_2\ ]\!](s, f)) & (10)
\end{array}
$$

Figure 3.3: Translating Geographic Data into CCS

- $\{\mathsf{in}_T(v) \mid T \in \mathcal{T}, v \in \{\texttt{c}, \texttt{o}\}\}$, representing minor cycle inputs from the track-side hardware. No assumption is made about the order in which these arrive, incoming values being simply copied to memory.

Note that Control is a serial recursive agent, and that the fanout from the Control state is exactly $\mathcal{Q} + \mathcal{U} + \mathcal{R} + 2\mathcal{T}$.

3.2.3 Translating Geographic Data into CCS

To complete the definition of the model a translation to CCS is provided for the grammar cited earlier. This translation is specified by the ten rules of the inductive definition displayed in Figure 3.3. The two syntactic forms of the one-armed conditional are given the same interpretation in rules (1) and (2). Note that in correspondence to Section 2.4, $\mathbf{T}[\![\cdot]\!]$ takes a pair of continuations as arguments—the first represents success of the rule, the second represents failure. $\mathbf{C}[\![\cdot]\!]$ carries only the successful continuation.

These rules need little comment, except in the treatment of the points test discussed in Section 2.4.2. When the "free to move" flag is present the test is disjunctive in rules (7) and (8), and if the points are in the wrong state the *PFM* data for the points are evaluated. In this case, since the other fields in the points memory are omitted in the model, we simply rewrite the test as shown.

An illustration of how the translation proceeds is worthwhile. For an example consider the panel request rule for the route R_{02}:

$$
\begin{array}{l}
\texttt{*Q02 if}\ \ P_1\,\texttt{crf}, T_1^{ac}\,\texttt{f}, T_2^{ab}\,\texttt{f}\\
\qquad\qquad \texttt{then}\ \ R_{02}\,\texttt{s}, P_1\,\texttt{cr}, T_1^{ca}\,\texttt{l}, T_2^{ba}\,\texttt{l}\ \ \texttt{\textbackslash .}
\end{array}
$$

If this code fragment is identified with the agent Q02, then

$$\mathsf{Q02} \;=\; \mathbf{C}[\![\mathit{PRR}(\texttt{*Q02})]\!]\mathsf{Control}$$

$$
\begin{array}{rcl}
\mathsf{Q02} & = & \mathbf{C}[\![\mathit{PRR}(\texttt{*Q02})]\!]\mathsf{Control} \\
 & = & \mathsf{get}_{P_1}(u).\mathit{if}\ (u = \texttt{cr})\ \mathit{then}\ \mathsf{Q02}_1\ \mathit{else}\ \mathsf{Q02}_2 \\
\mathsf{Q02}_1 & = & \mathbf{T}[\![T_1^{ac}\ \texttt{f}\,, T_2^{ab}\ \texttt{f}]\!](\mathsf{C02}, \mathsf{Control}) = \mathbf{T}[\![T_1^{ac}\ \texttt{f}]\!](\mathsf{Q02}_4, \mathsf{Control}) \\
 & = & \mathsf{get}_{T_1^{ac}}(u).\mathit{if}\ (u = \texttt{f})\ \mathit{then}\ \mathsf{Q02}_4\ \mathit{else}\ \mathsf{Control} \\
\mathsf{Q02}_2 & = & \mathbf{T}[\![T_1^{bc}\ \texttt{f}\,, T_1^{cb}\ \texttt{f}]\!](\mathsf{Q02}_1, \mathsf{Control}) = \mathbf{T}[\![T_1^{bc}\ \texttt{f}]\!](\mathsf{Q02}_3, \mathsf{Control}) \\
 & = & \mathsf{get}_{T_1^{bc}}(u).\mathit{if}\ (u = \texttt{f})\ \mathit{then}\ \mathsf{Q02}_3\ \mathit{else}\ \mathsf{Control} \\
\mathsf{Q02}_3 & = & \mathbf{T}[\![T_1^{cb}\ \texttt{f}]\!](\mathsf{Q02}_1, \mathsf{Control}) \\
 & = & \mathsf{get}_{T_1^{cb}}(u).\mathit{if}\ (u = \texttt{f})\ \mathit{then}\ \mathsf{Q02}_1\ \mathit{else}\ \mathsf{Control} \\
\mathsf{Q02}_4 & = & \mathbf{T}[\![T_2^{ab}\ \texttt{f}]\!](\mathsf{C02}, \mathsf{Control}) \\
 & = & \mathsf{get}_{T_2^{ab}}(u).\mathit{if}\ (u = \texttt{f})\ \mathit{then}\ \mathsf{C02}\ \mathit{else}\ \mathsf{Control} \\
\mathsf{C02} & = & \mathbf{C}[\![R_{02}\ \texttt{s}\,, P_1\ \texttt{cr}\,, T_1^{ca}\ \texttt{l}\,, T_2^{ba}\ \texttt{l}]\!]\mathsf{Control} \\
 & = & \overline{\mathsf{put}}_{R_{02}}(\texttt{s}).\overline{\mathsf{put}}_{P_1}(\texttt{cr}).\overline{\mathsf{put}}_{T_1^{ca}}(\texttt{l}).\overline{\mathsf{put}}_{T_2^{ba}}(\texttt{l}).\mathsf{Control}
\end{array}
$$

Figure 3.4: Panel route request `*Q02` translated into CCS

$$
\begin{array}{rcl}
 & = & \mathbf{T}[\![P_1\ \texttt{crf}\,, T_1^{ac}\ \texttt{f}\,, T_2^{ab}\ \texttt{f}]\!](\mathbf{C}[\![R_{02}\ \texttt{s}\,, P_1\ \texttt{cr}\,, T_1^{ca}\ \texttt{l}\,, T_2^{ba}\ \texttt{l}]\!]\mathsf{Control}, \mathsf{Control}) \\
 & = & \mathbf{T}[\![P_1\ \texttt{crf}]\!](\mathsf{Q02}_1, \mathsf{Control})
\end{array}
$$

by rules (1) and (9) respectively. $\mathsf{Q02}_1$ is a place holder—the control will reach this point if the points test succeeds, otherwise execution returns to the Control state. Continuing via rules (7) and (10):

$$
\begin{array}{rcl}
\mathbf{T}[\![P_1\ \texttt{crf}]\!](\mathsf{Q02}_1, \mathsf{Control}) & = & \mathbf{T}[\![\,(P_1\ \texttt{cr or}\ T_1^{bc}\ \texttt{f}\,, T_1^{cb}\ \texttt{f})\,]\!](\mathsf{Q02}_1, \mathsf{Control}) \\
 & = & \mathbf{T}[\![P_1\ \texttt{cr}]\!](\mathsf{Q02}_1, \mathsf{Q02}_2)
\end{array}
$$

$\mathsf{Q02}_2$ is a second place holder—execution continues from here (the "free to move" test) if the points are controlled normal instead of reverse. Finally

$$
\begin{array}{rcl}
\mathbf{T}[\![P_1\ \texttt{cr}]\!](\mathsf{Q02}_1, \mathsf{Q02}_2) & = & \mathsf{get}_{P_1}(u).\mathit{if}\ (u = \texttt{cr})\ \mathit{then}\ \mathsf{Q02}_1\ \mathit{else}\ \mathsf{Q02}_2 \\
 & = & \mathsf{get}_{P_1}(\texttt{cr}).\mathsf{Q02}_1 + \mathsf{get}_{P_1}(\texttt{cn}).\mathsf{Q02}_2
\end{array}
$$

by rule (6). Shown here is the result of the translation from value passing CCS syntax to the underlying calculus.

The rest of the clauses required in translating this panel request are given in Figure 3.4. This illustration completes the basic definition of the formal model of SSI, but it is not the last word we shall have to say about it. After discussing safety properties in the next section, it will be convenient to instrument Model #1 so that these can be formalised, and checked, appropriately. In Chapter 6 the model described above is extended with more of the apparatus of the SSI—in particular, input and output buffers, and watchdog timers.

3.3 Defining Safety Properties Formally

Technically, safety properties are associated with *invariants*. In sequential programs loop invariants are used to assert the correctness of while- and for-loops, for example; variant properties—*e.g.*, assertions that some program measure always diminishes—are used to make termination arguments. Termination need not be a concern here as there are no loop constructs in the language which could introduce infinite behaviour.

The safety property will be a formula **F** expressing logical relationships between the control variables of the SSI—a property which should hold after the execution of any of the rules in the database. That is to say it should be invariant under the state transformations induced by the Geographic Data. Formally, **F** is defined as a *modal* formula in the modal μ-calculus. We may prove the invariance of **F** by establishing the satisfaction relation:

$$(\mathsf{Control} \mid \mathsf{Image})\backslash L \models \nu Z.\mathbf{F} \wedge [-]Z$$

A state or process E satisfies the *temporal* formula $\nu Z.\mathbf{F} \wedge [-]Z$ if E satisfies **F** and every derivative E' of E satisfies $\nu Z.\mathbf{F} \wedge [-]Z$. This inductive definition of the notion of safety plays a crucial rôle in the development of the proof strategy in Chapter 4.

3.3.1 Safety Properties of Geographic Data

In Section 3.1 it was noted that it would not be possible to give formal proofs of *system* properties such as those listed drawn from the principles of railway signalling. Our thesis is that it is instead better to identify properties of the *data* since these can be formally verified with respect to a suitable model. If the model is valid the formal verification will give considerable weight to the necessarily informal argument that the safety principles have been adhered to in the interlocking's design. With this in mind the following properties are taken to be central to the correct design of the route request and sub-route release data:

MX It is never the case that two or more of the sub-routes over a given track section are simultaneously locked;

RT Whenever a route is set, all its component sub-routes are locked;

PT Whenever a sub-route over a track section containing points is locked, the points are controlled in alignment with that sub-route.

Throughout the sequel these three properties are referred to by the names **MX** (for mutual exclusion), **RT** (for routes), and **PT** (for points). Another property associated with points is also of interest:

PT The reverse (respectively, normal) sub-routes over points are free whenever the points are controlled normal (respectively, reverse).

If points are represented internally by binary variables the two formulations of **PT** are equivalent—in SSI, however, points do not have a binary representation so these are distinct properties.

Refining the above, one might wish to add the requirement that if a route is set the points along it are properly aligned—but this is unnecessary since **PT** and **RT** together ensure that the points along a route become correctly aligned when it is set. Drawing the example from the scheme plan on page 48, if R_{51} is set we require sub-routes T_6^{ac} and T_4^{ac} to be locked and the points P_3 and P_2 to be controlled normal and reverse respectively. If this is so, when the route is set the points will be correctly aligned; subsequently, if the route is unset, **PT** ensures that the points remain correctly aligned until the sub-route is freed.

Together, **MX** and **RT** are designed to ensure that no routes over the same section of track in opposing or otherwise conflicting directions are simultaneously set. Since signals that have routes associated with them are supposed to remain at red unless an onward route has been set, these properties ensure, in the absence of other faults, that no more than one train has access to any given track section at any one time. Naturally, that signals do have this property must also be verified—but this will arise from safety properties elsewhere in the data (*OPT* data, in particular). We shall see later in Section 4.4.3 that a slightly stronger form of **RT** is needed in general.

The foregoing discussion raises the question of *where* the invariant comes from. **MX** is easy enough to define given the naming conventions adopted for identifying track circuits and sub-routes. The other properties are more problematic. Using an informal notation for the present, and reading from the scheme plan on page 48, it is clear that the routes property for R_{51} should be:

$$R_{51}(\mathtt{s}) \Rightarrow T_6^{ac}(\mathtt{l}) \wedge T_4^{ac}(\mathtt{l}) \wedge T_2^{ab}(\mathtt{l})$$

However, that this is the correct relationship has to be taken 'on trust'—plainly it cannot be inferred from the data we wish to certify. In the absence of a generic characterisation of **RT** and **PT** it is therefore necessary to resort to specifying these by direct transcription from the scheme plan.

This raises the prospect that the invariant may be incorrect since transcription errors will likely arise if this is to be done by hand. Fortunately it is very much easier to define the invariant correctly than it is to compile the Geographic Data without error. Even so, in the worst case we should verify that the specified invariant is not inconsistent—*i.e.*, that it does not specify *false*. But it is easy to see that at least one model exists for the **F** if it expresses the above properties. Since **PT** and **RT** are implicative they will always

be true when the antecedent is *false*, or the consequent is *true*. Thus any assignment to the variables in which every route is unset and every sub-route is free will satisfy **PT** and **RT**: obviously it will also satisfy **MX**, and checking this is trivial.

3.3.2 Tags and Probes

Before discussing the precise definition of the formula **F** it is first necessary to repair a purely technical difficulty with Model #1. The problem is that one cannot observe the internal state of the model: the restriction set L precludes direct observation of the state of the variables in the Image of the railway. Nor is there any apparent mechanism by which the state of the system can be inferred from the actions that remain visible. Thus, although the model correctly expresses our intention *vis-à-vis* formal specification, it is of little worth from the verification standpoint. We remedy this situation with some technical machinery which, although having no functional purpose in itself, enables observation of the hidden states of the model.

Probes Walker used these in his work on mutual exclusion algorithms [97]. A probe may be used to observe an *action* hidden by the restriction operator; it is an action 'attached' to the hidden one, but which itself remains visible. In the model is is possible to probe $\overline{\mathsf{put}}_D(v)$, with an action like $\mathsf{obs}_D(v)$, so that whenever Control and Image synchronise this will be followed by an observation recording the event. By this means one may in principle keep track of the dynamic evolution of the image of the railway. There is a major drawback with the naïve use of probes however, for they introduce states to a *CCS* model that are entirely artificial. For $\mathsf{West}_{\#1}$, since there are so many probes needed, this presents a very serious overhead indeed. (It is worth noting that probes are much more natural in the analysis of synchronous models: in *SCCS*, the richer structure of actions means one may probe an action without introducing artificial states to the model.)

Tags Whereas probes are inserted in the execution sequence, tags are not intended for execution—they merely label some internal *state*. A tag is 'attached' to a state of interest by using sum, but it is easiest to explain the principle by an example. The registers given earlier are redefined thus:

$$\mathsf{Reg}_D(y) \stackrel{\text{def}}{=} \mathsf{put}_D(x).\mathsf{Reg}_D(x) + \overline{\mathsf{get}}_D(y).\mathsf{Reg}_D(y) + \mathsf{obs}_D(y).\mathsf{Reg}_D(y)$$

In this way one may always observe the state of the variable D, and hence the state of the image of the railway. In principle an external agent may interact with the model to ascertain the state of the system: moreover, such observation is quite transparent since the $\mathsf{obs}_D(y)$ actions neither introduce new states, nor change the current state of the

$$
\begin{array}{rcl}
\mathsf{Reg}_D(y) & \stackrel{\text{def}}{=} & \mathsf{put}_D(x).\mathsf{Reg}_D(x) + \overline{\mathsf{get}}_D(y).\mathsf{Reg}_D(y) + \mathsf{obs}_D(y).\mathsf{Reg}_D(y) \\
\mathsf{Image} & \stackrel{\text{def}}{=} & \prod_{U\in\mathcal{U}} \mathsf{Reg}_U(\mathtt{f}) \mid \prod_{R\in\mathcal{R}} \mathsf{Reg}_R(\mathtt{xs}) \mid \prod_{T\in\mathcal{T}} \mathsf{Reg}_T(\mathtt{c}) \mid \prod_{P\in\mathcal{P}} \mathsf{Reg}_P(\mathtt{cn}) \\
\mathsf{Control} & \stackrel{\text{def}}{=} & \mathit{Fix}(X.\,\mathsf{ctrl}.\mathbf{0} \\
& + & \sum_{Q\in\mathcal{Q}} \mathsf{set}_Q.(\mathbf{C}[\![\mathit{PRR}(\star\mathtt{Q})]\!]X) \\
& + & \sum_{U\in\mathcal{U}} (\mathbf{C}[\![\mathit{FOP}(U)]\!]X) \\
& + & \sum_{R\in\mathcal{R}} \mathsf{can}_R.\overline{\mathsf{put}}_R(\mathtt{xs}).X \\
& + & \sum_{T\in\mathcal{T}} \mathsf{in}_T(\mathtt{o}).\overline{\mathsf{put}}_T(\mathtt{o}).X \;+\; \mathsf{in}_T(\mathtt{c}).\overline{\mathsf{put}}_T(\mathtt{c}).X) \\
\mathsf{West} & \stackrel{\text{def}}{=} & (\mathsf{Control} \mid \mathsf{Image})\backslash L
\end{array}
$$

Model #2: An observable model of SSI

system. However, this is not the intention: we never expect Image to execute these new actions, we merely wish to exploit their presence in defining the modal formula **F**. In contrast, the tag in

$$\mathsf{Reg}_D(y) \stackrel{\text{def}}{=} \mathsf{put}_D(x).\mathsf{Reg}_D(x) + \overline{\mathsf{get}}_D(y).\mathsf{Reg}_D(y) + \mathsf{obs}_D(y).\mathbf{0}$$

introduces a new state, and a deadlock if the action is ever performed.

Observing SSI One reason for choosing the $\mathsf{obs}_D(y).\mathsf{Reg}_D(y)$ tag over the alternative is that it does not introduce new states to the model. This is important when one wishes to formally verify its properties (see Section 4.2). It turns out that it is also convenient to tag a particular state in the control. The state of interest is that between the end of one minor cycle and the beginning of the next. We therefore introduce the tag ctrl.**0** to Control, as displayed in Model #2 (the slightly baroque *Fix* notation is needed in the sequel). The reason for this minor intrusion is that the invariant need only hold at certain (safety critical) states in Control. This is made clear in the next section. Note that this tag *does* introduce a new state: in fact, it doubles the state space of the model (this can be avoided using the tag ctrl.X as described in Section 3.3.4 below.) In Model #2 note that $L = \{\mathsf{put}_D(v), \mathsf{get}_D(v) \mid D \in \mathcal{D}\}$.

3.3.3 Geographic Data Invariants

The properties discussed in Section 3.1 are easily expressed as state properties in the modal μ-calculus. For **MX**, for example, and for the track section T_2 in particular, any state in which both sub-routes are locked will satisfy the formula:

$$\langle \mathsf{obs}_{T_2^{ab}}(\mathtt{l})\rangle \mathit{tt} \wedge \langle \mathsf{obs}_{T_2^{ba}}(\mathtt{l})\rangle \mathit{tt}$$

$\mathbf{RT} \stackrel{\text{def}}{=} \bigwedge$		$\mathbf{MX} \stackrel{\text{def}}{=} \bigwedge$		$\mathbf{PT} \stackrel{\text{def}}{=} \bigwedge$
$\mathbf{RT}(R_{02}, [T_1^{ca}, T_2^{ba}])$		$\mathbf{MX}[T_0^{ab}, T_0^{ba}]$		$\mathbf{PT}_{\mathtt{cn}}(P_1, [T_1^{bc}, T_1^{cb}])$
$\mathbf{RT}(R_{04}, [T_1^{cb}, T_3^{ba}])$		$\mathbf{MX}[T_2^{ab}, T_2^{ba}]$		$\mathbf{PT}_{\mathtt{cr}}(P_1, [T_1^{ac}, T_1^{ca}])$
$\mathbf{RT}(R_1, [T_1^{ac}, T_0^{ab}])$		$\mathbf{MX}[T_3^{ab}, T_3^{ba}]$		$\mathbf{PT}_{\mathtt{cn}}(P_2, [T_4^{ab}, T_4^{ba}])$
$\mathbf{RT}(R_3, [T_1^{bc}, T_0^{ab}])$		$\mathbf{MX}[T_5^{ab}, T_5^{ba}]$		$\mathbf{PT}_{\mathtt{cr}}(P_2, [T_4^{ac}, T_4^{ca}])$
$\mathbf{RT}(R_5, [T_6^{ab}, T_5^{ab}])$		$\mathbf{MX}[T_7^{ab}, T_7^{ba}]$		$\mathbf{PT}_{\mathtt{cn}}(P_3, [T_6^{ac}, T_6^{ca}])$
$\mathbf{RT}(R_6, [T_6^{ba}, T_7^{ba}])$		$\mathbf{MX}[T_1^{ac}, T_1^{bc}, T_1^{ca}, T_1^{cb}]$		$\mathbf{PT}_{\mathtt{cr}}(P_3, [T_6^{ab}, T_6^{ba}])$
$\mathbf{RT}(R_2, [T_4^{ca}, T_6^{ca}, T_7^{ba}])$		$\mathbf{MX}[T_4^{ab}, T_4^{ac}, T_4^{ba}, T_4^{ca}]$		
$\mathbf{RT}(R_4, [T_4^{ba}, T_6^{ca}, T_7^{ba}])$		$\mathbf{MX}[T_6^{ab}, T_6^{ac}, T_6^{ba}, T_6^{ca}]$		
$\mathbf{RT}(R_{51}, [T_6^{ac}, T_4^{ac}, T_2^{ab}])$				
$\mathbf{RT}(R_{53}, [T_6^{ac}, T_4^{ab}, T_3^{ab}])$				

Figure 3.5: Geographic Data invariant for WEST

The invariant required is therefore the logical negation of this:

$$\mathbf{MX}[T_2^{ab}, T_2^{ba}] \stackrel{\text{def}}{=} [\mathsf{obs}_{T_2^{ab}}(\mathtt{l})]ff \vee [\mathsf{obs}_{T_2^{ba}}(\mathtt{l})]ff$$

A similar term will express the same condition for each of the other track sections in the interlocking. In sections containing a points switch the sub-routes have to be taken pair-wise, giving rise to (up to) six conjuncts of disjuncts such as this.

Properties **RT** and **PT** may be similarly characterised. For route R_{02} and the points along it the following are required to hold invariantly:

$$\begin{aligned}
\mathbf{RT}(R_{02,}[T_1^{ca}, T_2^{ba}]) &\stackrel{\text{def}}{=} \langle \mathsf{obs}_{R_{02}}(\mathtt{s})\rangle tt \Rightarrow \langle \mathsf{obs}_{T_1^{ca}}(\mathtt{l})\rangle tt \wedge \langle \mathsf{obs}_{T_2^{ba}}(\mathtt{l})\rangle tt \\
\mathbf{PT}_{\mathtt{cr}}(P_1, [T_1^{ac}, T_1^{ca}]) &\stackrel{\text{def}}{=} \langle \mathsf{obs}_{T_1^{ac}}(\mathtt{l})\rangle tt \vee \langle \mathsf{obs}_{T_1^{ca}}(\mathtt{l})\rangle tt \Rightarrow \langle \mathsf{obs}_{P_1}(\mathtt{cr})\rangle tt \\
\mathbf{PT}_{\mathtt{cn}}(P_1, [T_1^{bc}, T_1^{cb}]) &\stackrel{\text{def}}{=} \langle \mathsf{obs}_{T_1^{bc}}(\mathtt{l})\rangle tt \vee \langle \mathsf{obs}_{T_1^{cb}}(\mathtt{l})\rangle tt \Rightarrow \langle \mathsf{obs}_{P_1}(\mathtt{cn})\rangle tt
\end{aligned}$$

For the record, Figure 3.5 defines the required safety property for WEST. To avoid burdensome notation in the sequel we shall usually use abbreviations **MX**, **RT**, $\mathbf{PT}_{\mathtt{cn}}$ and $\mathbf{PT}_{\mathtt{cr}}$ as above.

The safety property for WEST is therefore taken to be the conjunctive term $\mathbf{F} \stackrel{\text{def}}{=} \mathbf{MX} \wedge \mathbf{RT} \wedge \mathbf{PT}$ in Figure 3.5. Although a large formula, this in itself is not problematic for model checking algorithms which automate the invariance proof—the space complexity of the model is by far the greatest obstacle. Still, the proof obligation we require to discharge is now fully defined with the property **F**, given above, and Model #2:

$$(\mathsf{Control} \mid \mathsf{Image})\backslash L \models \nu Z.\mathbf{F} \wedge [-]Z$$

It turns out, however, that this analysis is too fine: **F** does not hold in every state of the model as required to satisfy this formula. This becomes apparent when considering the

route request rule for R_{02} (see Figure 3.4). During the computation several states are encountered in which the route is only partly locked (in C02), so **PT** and **RT** will not generally hold until all of the variables have been updated in setting the route.

However, the control is a sequential machine and SSI never takes decisions based on the values of the variables in these intermediate (or transient) states—they are never evaluated by the guard in a command. This leads to the observation that the safety critical states of the system are those in which the SSI is about to evaluate such a guard. We therefore conclude that the invariant can be weakened, using the ctrl tag, to:

$$\mathbf{\Phi} \stackrel{\text{def}}{=} \nu Z.(\langle \mathsf{ctrl} \rangle tt \Rightarrow \mathbf{F}) \wedge [-\mathsf{ctrl}]Z$$

That is, **F** need only hold at the tagged control states. The $[-\mathsf{ctrl}]$ modality ensures that only those runs through the state space that do not follow the ctrl action, which leads to the deadlocked control, are considered in the proof.

3.3.4 Generalising the Translation Schema

In Model #2 the labelled states of the Control represent the beginning (or end) of each minor cycle. However, the observation that it is these that are the safety critical states is only strictly valid for the guarded command language assumed here: when guarded commands are placed in sequence it is necessary to be more careful about identifying the critical states of the model.

In sketching how these ideas can be generalised, we first extend the grammar given in Figure 3.2 with a new phrase form:

$$\begin{aligned} \langle \text{gd} \rangle &::= \langle \text{prr} \rangle \mid \langle \text{fop} \rangle \mid \langle \text{gd} \rangle \texttt{ ; } \langle \text{gd} \rangle \\ \langle \text{prr} \rangle &::= \texttt{if } \langle \text{tl} \rangle \texttt{ then } \langle \text{cl} \rangle \texttt{ else } \langle \text{cl} \rangle \end{aligned}$$

(this syntax is not correct Geographic Data, but it serves to illustrate the principle). Then the appropriate rule in the translation to CCS is supplied which identifies the safety critical states in the process:

$$\begin{aligned} \mathbf{C}[\![\langle \text{gd} \rangle_1 \texttt{ ; } \langle \text{gd} \rangle_2]\!]c &= \mathbf{C}[\![\langle \text{gd} \rangle_1]\!](\mathsf{ctrl}.\mathbf{0} + \mathbf{C}[\![\langle \text{gd} \rangle_2]\!]c) \\ \mathbf{C}[\![\texttt{if } \langle \text{tl} \rangle \texttt{ then } \langle \text{cl} \rangle_1 \texttt{ else } \langle \text{cl} \rangle_2]\!]c &= \mathbf{T}[\![\langle \text{tl} \rangle]\!](\mathbf{C}[\![\langle \text{cl} \rangle_1]\!]c, \mathbf{C}[\![\langle \text{cl} \rangle_2]\!]c) \end{aligned}$$

There are alternatives to this scheme of course, but they will not be explored far here. In the sequel we suppose that the safety property expressed in **F** should hold at the intermediate control points, so the invariant is defined just as it was above, and we use the ctrl.**0** tag throughout.

In order to avoid introducing artificial states to the model, with ctrl.**0**, one could instead translate sequence thus

$$\mathbf{C}[\![\langle \text{gd} \rangle_1 \texttt{ ; } \langle \text{gd} \rangle_2]\!]c = \mathbf{C}[\![\langle \text{gd} \rangle_1]\!](\mathit{Fix}\,(Y.\,\mathsf{ctrl}.Y + \mathbf{C}[\![\langle \text{gd} \rangle_2]\!]c))$$

as long as Y is not free in $\mathbf{C}[\![\, \langle \mathrm{gd} \rangle_2 \,]\!]c$. On the other hand one may be satisfied with a weaker property holding at the intermediate control points—in which case a different tag could be used and the invariant modified appropriately. For example, $\mathbf{\Phi}$ can be generalised along these lines:

$$\nu Z.(\langle \mathsf{c}_1 \rangle tt \Rightarrow \mathbf{F}_1 \wedge \langle \mathsf{c}_2 \rangle tt \Rightarrow \mathbf{F}_2 \wedge \langle \mathsf{c}_3 \rangle tt \Rightarrow \mathbf{F}_3) \wedge [-\mathsf{c}_1, \mathsf{c}_2, \mathsf{c}_3]Z$$

where c_1, c_2 and c_3 are distinct control tags.

3.4 The Problem with State Spaces

The space complexity of the model presented is (potentially) enormous although much of the behaviour has been abstracted and the example is itself rather small. Yet of the $2^{|\mathcal{D}|}$ possible configurations of Image, $|\mathcal{D}|$ being typically in the hundreds, only a tiny percentage are expected to be reachable from any given initial state. Given an efficient representation we therefore expect model checking techniques could prove the invariant. Efficiency is needed both in constructing the model, and in its storage. Firstly, however, the model's space complexity can be substantially reduced through the application of some *agent transformations* [72, 73].

3.4.1 Hiding Assumptions

The first transformation in simplifying the model is to *hide* some of the observable actions. Hiding does not preserve the observable behaviour in an agent of course, but focuses the analysis instead on particular aspects of the model. A simple example of hiding irrelevant behaviour would be to hide the tags introduced in Model #2. Suppose $O = \{\mathsf{obs}_D(v) \mid D \in \mathcal{D}\} \cup \{\mathsf{ctrl}\}$. If Model #2 is defined using only the non-deadlocking tags then $\mathsf{West}_{\#2} \,/O = \tau.\mathsf{West}_{\#1}$. This follows by a straightforward application of the τ-laws, and the Recursion Laws (in particular $\mathit{Fix}\,(X.\,\tau.X + E) = \mathit{Fix}\,(X.\,\tau.E)$) for observation congruence [62].

Hiding can be used to prove the *absence* of deadlock in a model by appealing to the intuition that deadlocks do not arise from the observable actions of a system [73]. Hiding is used here for a different purpose: to abstract the behaviour associated with track circuits. This is justified by appealing to the following premises:

- *jumping trains axiom*: we assume track circuits may change state unconditionally at any time;
- *unconditional route cancellation*: we assume that routes may be cancelled unconditionally at any time.

The jumping trains axiom (for want of a better name!) has some interesting consequences: firstly it is clear that space complexity of the model is proportional to $2^{|\mathcal{T}|}$; secondly, as explained below, if the initial actions of the Input component are hidden the state space can be reduced by this factor; thirdly, adopting this premise negates any possibility of examining safety properties of signal data. (Signal aspects are tightly interlocked with the state of the tracks in advance and in the rear.) It is for this reason that the jumping trains axiom is not an assumption which is intrinsic to the model.

Although there may be some doubt as to the validity of the jumping trains axiom, it is clear that in the unlikely presence of faulty track circuiting in the railway, track circuits may indeed appear to the SSI as if they change state at random. This is, in fact, the weakest assumption that can be made about the behaviour of the environment in which the SSI operates—that is, we make no assumption at all. Consequently, any safety properties of the Geographic Data that can be established under these conditions will be robust indeed, being certainly enjoyed by the SSI when placed in a more orderly environment.

We shall therefore temporarily adopt the two premises above in verifying safety properties of the route request and sub-route release data. In doing so, we might as well hide the behaviour associated with these aspects of the model—in particular the visible actions in $C = \{\mathsf{can}_R \mid R \in \mathcal{R}\}$ and $I = \{\mathsf{in}_T(v) \mid T \in \mathcal{T}, v \in \{\mathtt{c}, \mathtt{o}\}\}$.

3.4.2 Agent Transformations

Using the *Fix* notation introduced earlier we proceed with the untagged Model #1 and let $\mathsf{Control} = \mathit{Fix}\,(X.\,\mathit{Sum} + \mathit{Input})$, where *Input* abbreviates the Input behaviour, and *Sum* abbreviates the rest. The agent

$$\begin{aligned}
\mathsf{West}_{\#1}\,/I &= (\mathsf{Control}\,/I \mid \mathsf{Image})\backslash L \\
&= (\mathit{Fix}\,(X.\,\mathit{Sum}\,/I + \mathit{Input}\,/I) \mid \mathsf{Image})\backslash L \\
&= (\mathit{Fix}\,(X.\,\mathit{Sum} + \sum_{T\in\mathcal{T}} \tau.\overline{\mathsf{put}}_T(v).X) \mid \mathsf{Image})\backslash L
\end{aligned}$$

by the static laws since the actions in I occur only in *Input*.

In itself, hiding these visible actions achieves no compression in the state space. To do that we first have to use the idea of *local expansion*. Consider:

$$A \stackrel{\text{def}}{=} (\mathsf{Control}\,/I \mid \mathsf{Reg}_T(\mathtt{o}))\backslash L_T$$

where $L_T = \{\overline{\mathsf{get}}_T(\mathtt{o}), \overline{\mathsf{get}}_T(\mathtt{c}), \mathsf{put}_T(\mathtt{o}), \mathsf{put}_T(\mathtt{c})\}$. This, for any particular $T \in \mathcal{T}$, is a local component of $\mathsf{West}\,/I$ because $L_T \subseteq \mathcal{L}(\mathsf{Image})$ and only these two agents communicate over the actions in L_T. The idea is to apply the Expansion Law to arrive at a serial recursive agent that is equal to the pair. Taking observation equivalence as

the notion of equality it is possible to further abstract the silent actions appearing in the expansion; then partitioning the state space with respect to $\approx$ we derive a minimal serial agent equivalent to A.

The question then is how 'minimal' this transformed agent is compared with A. In this case the heuristics work favourably. If $\mathsf{Control}$ has K states then $\mathsf{Control}\,/I$ also has K states and clearly A has $2K$ states as the register is binary. However, it is not difficult to see that A is observation equivalent to $B \stackrel{\text{def}}{=} (\mathsf{Control}\,/I \mid \mathsf{Reg}_T(\mathtt{c}))\backslash L_T$. Indeed, the agents are congruent. In detail, for the congruence the initial transitions of each agent must be matched with at least one transition of the other:

$$\left.\begin{array}{lll} A & \xrightarrow{\tau} & (\overline{\mathsf{put}}_T(v).\mathsf{Control}\,/I \mid \mathsf{Reg}_T(\mathtt{o}))\backslash L_T \\ B & \xrightarrow{\tau} & (\overline{\mathsf{put}}_T(v).\mathsf{Control}\,/I \mid \mathsf{Reg}_T(\mathtt{c}))\backslash L_T \end{array}\right\} \approx \qquad \text{(i)}$$

$$\left.\begin{array}{lll} A & \xrightarrow{\alpha} & (C'\,/I \mid \mathsf{Reg}_T(\mathtt{o}))\backslash L_T \\ B & \stackrel{\alpha}{\Longrightarrow} & (C'\,/I \mid \mathsf{Reg}_T(\mathtt{o}))\backslash L_T \end{array}\right\} \equiv \qquad \text{(ii)}$$

$$\left.\begin{array}{lll} B & \xrightarrow{\alpha} & (C'\,/I \mid \mathsf{Reg}_T(\mathtt{c}))\backslash L_T \\ A & \stackrel{\alpha}{\Longrightarrow} & (C'\,/I \mid \mathsf{Reg}_T(\mathtt{c}))\backslash L_T \end{array}\right\} \equiv \qquad \text{(iii)}$$

A suitable (congruence) relation therefore consists of the above pairs together with the identity relation. In (i) the pairs are bisimilar since the only derivative in each case is $(\mathsf{Control}\,/I \mid \mathsf{Reg}_T(v))\backslash L_T$; in (ii) and (iii) C' may be any (other) intermediate state of $\mathsf{Control}$ reachable in one step (or several steps by first resetting the register in the simulating agent). This relation is not minimal, but note that

$$(C'\,/I \mid \mathsf{Reg}_T(\mathtt{o}))\backslash L_T \quad \text{and} \quad (C'\,/I \mid \mathsf{Reg}_T(\mathtt{c}))\backslash L_T$$

are observation equivalent whenever C' is such that however it returns to the $\mathsf{Control}$ state it does so without interacting, via $\overline{L_T}$, with the register. This is a sufficient condition, and the graph partitioning algorithm implemented in the Edinburgh Concurrency Workbench [20] will identify such pairs in finding a suitable bisimulation. It follows that the minimal serial agent that is observation equivalent to A also has (approximately) $|\mathsf{Control}| = K$ states.

Note that to arrive at this result it is necessary to hide the I actions for otherwise $A \not\approx B$. As a corollary to the above

$$(\mathsf{Control} \mid \prod_{T\in\mathcal{T}} \mathsf{Reg}_T(v))\backslash L_{\mathcal{T}}$$

where $L_{\mathcal{T}} = \{\mathsf{put}_T(v), \overline{\mathsf{get}}_T(v) \mid T \in \mathcal{T}, v \in \{\mathtt{c},\mathtt{o}\}\}$ clearly has $2^{|\mathcal{T}|}K$ states—so the model has space complexity exponential in the number of track circuits. Hiding the input actions therefore factors out the $2^{|\mathcal{T}|}$.

The actions $C = \{\mathsf{can}_R \mid R \in \mathcal{R}\}$ can also be hidden, now in the agent $\mathsf{West}_{\#1}\,/I$, but on this occasion the heuristics achieve little further compression in the model. The same transformation works for Model #2, but the congruence proof needs a minor change since the right-hand agents at (i) are not then bisimilar.

3.4.3 Model Checking

Following the simplifications above, the question is whether it is possible now to use the Concurrency Workbench to establish $\mathsf{West}/I \models \mathbf{\Phi}$? Model checking algorithms to decide this question come in two varieties, global and local (the tool provides both), so a brief comparison is useful.

Global Model Checking Here, a property Φ is shown to hold at a state s in a model $\mathcal{M}$ by first enumerating $\mathcal{M}$'s states, finding the set of these for which the property holds, and checking that s is in the set. This, at least, is the approach advanced by Clarke, Emerson and Sistla [18]. Their models are Kripke structures (*unlabelled* transition systems) and the property language is Computation Tree Logic (CTL). In this language the invariant $\mathbf{\Phi}$ can be expressed by the temporal formula $\mathbf{AG}\,(C \Rightarrow \mathbf{F})$, where C is an atomic proposition true at the control states, and $\mathbf{F}$ has been suitably recoded (along all paths globally, C implies $\mathbf{F}$). States of the model are just vectors of variables like Image. Note that to identify (tag) the control states an additional variable is needed in the representation. Clarke's model checker has been adapted for use with process algebraic models directly [31]—but, as with the Concurrency Workbench, inefficiency in the algorithm used to construct the model is the first obstacle to overcome in applying these tools (*cf.* Section 3.5).

Burch, McMillan and others [13, 57] claim good results for an approach to model checking CTL formulae that adapts Clarke's global algorithm in another way. Their methods use a compact representation for Boolean formulae, known as binary decision diagrams [12], to represent the state space symbolically—as opposed to explicitly, by the graph. McMillan's system uses a richer logic, the propositional μ-calculus, to encode the transition relation of the model: the transition system is then represented by the transitive closure of this relation. It is the computation of the transitive closure that presents the most serious limitation of this technology because, when measured in terms of the number of nodes needed in the binary decision diagram, the intermediate computations can far exceed the space required to represent the model. Heuristics to alleviate these difficulties have been found to help for particular circuits [14], but they are not generally reliable.

Symbolic model checking is certainly useful in hardware verification, but the applicability to the world of CCS models is uncertain. Taubner *et al.* [30] show that the representation of the product of N transition relations requires a binary decision diagram of size $\mathrm{O}(2^{|Act|} \cdot \sum_{i=1}^{N} |S_i|^2)$, where $|S_i|$ is the size of each parallel component. This is a good bound when the signature is fixed and it is the degree of parallelism that varies. But for Model #2 this bound is $\mathrm{O}(2^{|Act|} \cdot (4N + K^2))$. Doubling the size of the problem—*i.e.*, the interlocking, which doubles the number of registers and rules—

doubles all of the parameters Act, N and K, so giving at least an eightfold increase in the size of the representation of the model. These are asymptotic bounds, but precise estimates are notoriously difficult where binary decision diagrams are concerned.

Local Model Checking On the other hand one can demonstrate that Φ holds at a state s in a model $\mathcal{M}$ by a purely local argument since one need only explore $\mathcal{M}$ in the neighbourhood of the (initial) state of interest: often this is enough to demonstrate that Φ holds at s, and avoids construction of the entire model. So from a practical point of view the key idea behind local model checking is that it is a lazy proof method. The tableau proof technique due to Stirling and Walker [90] is lazy in this sense, the proof rules used to construct the tableau being governed by the actions in the modalities: these specify the minimal set of next states to consider in traversing the transition system. This model checker is described in more detail in Appendix B.

A major difficulty arises with the tableau proof method described by Stirling and Walker since the algorithm has very poor (*i.e.*, exponential) worst case complexity [91]. The algorithm implemented by Cleaveland [19] in the Concurrency Workbench is more efficient, running in time polynomial in the size of the model for a formula as simple as $\mathbf{\Phi}$, but this also has poor worst case performance. Andersen [2, 3] describes a *global* algorithm for the modal μ-calculus that, for a class of formulae including $\mathbf{\Phi}$ and all CTL formulae, runs in time linear in the size of the structure and the formula. (This is the same complexity as Clarke's CTL model checker.) Andersen's *local* version, for the same class of formulae, runs in time that is no more than a log factor worse than the global version. The worst case performance of these local algorithms becomes important if, due to the structure of the particular model, or formula, the algorithm has to do what the global model checkers do: examine the global state space of the model.

Local model checking is most useful when one is interested in checking a property that involves only a fraction of the actions an agent can perform. Liveness and fairness properties are often good candidates here, but global invariants such as $\mathbf{\Phi}$ are not. Local model checking is also of practical interest when the models are infinite (where global methods are obviously inadequate). Bradfield and Stirling [10] have extended the tableau proof technique for the modal μ-calculus to a semi-automatic method for infinite models—but a formula such as $\mathbf{\Phi}$ would be dealt with by the automatic part of the algorithm, which is also global in character.

Model Checking Geographic Data Returning, then, to the question of whether $\mathsf{West}/I \models \mathbf{\Phi}$, it appears that local model checking offers no practical advantage since the structure of the invariant necessarily entails exploring the entire state space—this arises from the box modality, $[-\mathsf{ctrl}]$, which only excludes transitions labelled by the

ctrl action in building the model. Even so, for WEST, the global model checker in the Concurrency Workbench answers the question affirmatively, although one has to be content to follow the unorthodox path of reprogramming the tool to handle this class of CCS models more efficiently than the generic algorithms are capable of. Moreover, the local model checker is effectual when the property Φ does not hold of the model because the algorithm terminates with a negative result while the transition system is only partially built, but a mechanism to generate counterexamples would make the tool more useful as a debugging aid than in its present incarnation.

In general however, even with the jumping trains abstraction, the number of reachable safe states quickly becomes astronomical. Space complexity in the problem of checking safety properties of Geographic Data is endemic, presenting a severe challenge for verification methods based on model construction. Even so, in the next chapter we shall see that Stirling and Walker's tableau method for the modal μ-calculus reveals a simple inductive proof method which is linear in the number of rules in the data, and independent of the number of states of the system. Chronologically however, the next stage in the analysis focused on the problem of efficiently generating the state space of the model described in here. This is the subject of the brief digression from the formal analysis described next.

3.5 Proof by Program

One difficulty encountered in trying to analyse the present model of SSI in an automated tool arises in the method by which the model is constructed from the textual description. As long as the goal remains the direct exploration of the states space, we should supply the intelligence that the Control is the dynamic component, while the Image is passive. This leads to an efficient means of generating the transition system.

3.5.1 Generating States of SSI

One way of constructing the model is to compute the product automaton from the component parts whose specifications are given. For the present model this would immediately run into trouble because the Image component really has $2^{|\mathcal{D}|}$ states—the restriction is only applied at the outermost level. Since there is no communication between any of the Image components we can improve the naïve method considerably by applying a smaller restriction (smaller than $\backslash L$) at each step in constructing $(((\mathsf{Control} \mid I_1)\backslash L_1 \mid I_2)\backslash L_2 \mid \cdots)\backslash L_n$. But this will not help in general either as it gives rise to intermediate representations which may be far too large to handle because the smaller restrictions (sets L_1, L_2, *etc.*) permit more behaviour than is intrinsic to the model.

$$\text{*Q02}_{cr}\ \frac{P_1 \in \overline{N} \qquad \{T_1^{ac}, T_2^{ab}\} \subseteq F}{(N,S,F) \rightsquigarrow (N, S \cup \{R_{02}\}, F - \{T_1^{ca}, T_2^{ba}\})}$$

$$\text{*Q02}_{cn}\ \frac{P_1 \in N \qquad \{T_1^{ac}, T_1^{bc}, T_1^{cb}, T_2^{ab}\} \subseteq F}{(N,S,F) \rightsquigarrow (N - \{P_1\}, S \cup \{R_{02}\}, F - \{T_1^{ca}, T_2^{ba}\})}$$

$$\frac{R_{02} \in \overline{S}}{(N,S,F) \rightsquigarrow (N,S,F \cup \{T_1^{ca}\})} \qquad \frac{T_1^{ca} \in F}{(N,S,F) \rightsquigarrow (N,S,F \cup \{T_2^{ba}\})}$$

$$\frac{}{(N,S,F) \rightsquigarrow (N, S - \{R\}, F)}$$

Figure 3.6: Transition rules for *PRR* and *FOP* data

Instead, a program is developed to generate the safety critical states which is based directly on a simple transitional semantics (formally justified later in Section 4.3). We discard the transition information since our interest is not currently in the structure of the automaton induced by the rules in the Geographic Data, but in its states. These are then checked against the invariant, suitably encoded.

Semantics The states of the machine are represented by three sets (N, S, F), where $N \subseteq \mathcal{P}$ is the collection of points that are controlled *normal*, $S \subseteq \mathcal{R}$ is the collection of routes that are *set*, and $F \subseteq \mathcal{U}$ is the collection of sub-routes that are *free*. Each rule in the route request and sub-route release data defines a transition between the set of states satisfying the test in the rule, to a new set of states resulting from the assignments in the rule's command. This unlabelled transition relation is represented by $\rightsquigarrow$, and is defined by interpreting the *PRR* and *FOP* data as per the examples in Figure 3.6.

There will generally be several rules for each panel route request due to the disjunctive test on points. The last of the rules in Figure 3.6 specifies that routes may be cancelled unconditionally at any time. Track circuits have been removed from consideration from the system, as in Section 3.4, so they do not appear in the state or in the sub-route release rules. Also, if a rule does not apply in some state then no action is taken—because if a rule fails in SSI, the image of the railway goes unmodified.

Compilation The *PRR* and *FOP* files have to be read and the data compiled into some suitable internal representation. During parsing there is an opportunity to carry out some syntactic analysis on the data. For example, by exploiting the naming conventions employed in the identity files it is possible to check that:

- the points command in the conclusion of a route request rule corresponds to the points tested in the condition;

- the track circuit tested in a sub-route release rule corresponds to the sub-route being freed;
- that the sub-routes appearing in the *PFM* data occur in opposing pairs (there are exceptions where tracks are unidirectional) over the same track circuit;

and so on. One can also find sub-route 'chains' in the *FOP* data: each rule should test the immediately preceding sub-route(s), unless it is the rule for the first sub-route in which case it should test the route variable(s). The *maximal* chains—roughly speaking, the closure of the 'chain' relation—in the *FOP* data should correspond to the sub-routes locked in the *PRR* data (in some order). While these syntactic checks can remove many typographic errors, they do not help the behavioural analysis except in so far as we can assume data submitted for behavioural analysis pass such tests. (It is worth noting that deliberate errors in the FOREST LOOP data [8, see page 215] received from British Rail did not even pass this syntactic analysis.)

Data Structures Boolean vectors are a natural, compact representation for the states of the SSI. Given the right programming language, functions such as complement and bit-wise "and" and "or" may be implemented highly efficiently. Each state is therefore represented by a Boolean vector $\mathbf{v}$ of fixed length $d = |\mathcal{D} - \mathcal{T}|$. Given a fixed ordering on $\mathcal{P} \uplus \mathcal{R} \uplus \mathcal{U}$ one can represent D_j by the vector $\mathbf{d}_j$, also of length d, having all but the j^{th} bit clear. The interpretation is that if the j^{th} bit is set in state $\mathbf{v}$ the variable represented by $\mathbf{d}_j$ is in the state *normal*, *set* or *free*, depending on the partition ($\mathcal{P}$, $\mathcal{R}$ or $\mathcal{U}$) to which it belongs.

The finite set operations appearing in the transition rules, like member, union, and subset, are implemented in logical operations lifted to Boolean vectors, taking equality on Boolean vectors to be primitive—*e.g.* the equality on the underlying (implementation) type. Set difference, $V - X$, is encoded as $\overline{\mathbf{x}} \cdot \mathbf{v}$, and $D_j \in V$ as $\mathbf{d}_j \cdot \mathbf{v} = \mathbf{d}_j$, for example. (Dot product represents bit-wise "and", $\overline{\mathbf{x}}$ represents ones complement, and we use juxtaposition to represent "or".) Then a fast data structure is needed to hold sets of states. Threaded binary trees, for the minimal overhead of two additional bits for the threads (as well as the pointers to subtrees), offer a representation for which, in the average case, insertion and deletion operations have logarithmic time complexity. We use the natural ordering on the data (the order on the integers $[0, 2^d - 1]$) and may approached the average case complexity by randomly ordering $\mathcal{P} \uplus \mathcal{R} \uplus \mathcal{U}$.

Algorithm Since the task is not that of building the transition graph, only that of enumerating the states of SSI, the algorithm to generate the state space is brutally simple. The transition rules define a function which takes a state to a set of states;

the application of this function is therefore iterated over all (reachable) states of the SSI. Starting from some set of (presumably safe) initial states I_0 (such as $\{\mathbf{0}\}$) the first iteration generates its immediate descendants I_1. Let $I_1 \leftarrow I_1 - I_0$ be the generating set on the next iteration, and update $I_0 \leftarrow I_1 \cup I_0$ with the new states generated. The process terminates when the generating set is empty, and in this way all states reachable from the initial state(s) are generated. The time complexity depends on the number of transition rules and on the number of reachable states of course; but the space complexity of the algorithm, and indeed of railway interlockings in general, is difficult to analyse formally as the precise relationship between d, the number of rules, and the number of reachable states is highly obscure (no useful measure has emerged from the parameters and figures in Figure 3.7 at least).

The above method can be easily adapted to generate the transition graph, but then memory requirements become seriously limiting because of the need to represent a large number of transitions. The algorithm can also be extended to account for track circuits, but then the state space, as already indicated, is exponential in their number. But when track circuits *are* added to the state (and one is not interested in constructing the state space of the SSI) the programs discussed here may be readily integrated into a simulation environment for the purpose of testing the interlocking design.

3.5.2 Checking Properties

Once the states of the model have been enumerated, the global safety analysis can proceed since it is required only to verify that each state satisfies the given safety property. For instance, for route r over sub-routes a, b and c, $\mathbf{RT}(r, [a, b, c])$ is satisfied by a state (N, S, F) if $r \in \overline{S}$, or if $\{a, b, c\} \subseteq \overline{F}$. This may be expressed as a disjunctive test on the Boolean vector $\mathbf{v}$ representing the state: $\mathbf{r} \cdot \mathbf{v} = \mathbf{0}$ *or* $(\mathbf{abc}) \cdot \overline{\mathbf{v}} = (\mathbf{abc})$, for example. Generally, since $\mathbf{F}$ is a predicate on states it is necessary to check $\mathbf{F}(\mathbf{v})$ is true for each reachable state $\mathbf{v}$.

Results WEST is a very small SSI and can be constructed and analysed by the methods indicated above in a matter of seconds. THORNTON JN. (see page 216) is obviously more complex, having six sets of points and twice as many rules. The table in Figure 3.7 summarises the results of the experiment, though timings are only approximate (for a Sun4 workstation). In conducting these experiments numerous typographic errors in the data were exposed by the syntactic analysis; only one (unseeded) semantic error passed through to the verification stage before being revealed.

The strategy of constructing the model before trying the verification step is not optimal (just easier to describe). Instead, the two programs are combined so that at each iteration of the generation algorithm the generating set is immediately checked

INTERLOCKING	$(\mathcal{P},\mathcal{R},\mathcal{U})$	$(\mathcal{T})$	Size	Genesis	Analysis
WEST	(3,10,22)	(8)	5,072	8 s	1 s
EAST-WEST	(4,14,32)	(12)	165,856	17 m	56 s
FOREST LOOP	(4,16,32)	(12)	695,552	2 h 50 m	6 m
THORNTON JN.	(6,16,40)	(14)	1,373,532	7 h 40 m	22 m

Figure 3.7: Results of Proof by Program

for errors. **MX** can be checked as each new state is added to the graph. Seeding the data with errors it was found that they were always revealed within the first few iterations of the program. Nevertheless, to verify that there are *no* errors one has to examine the entire state space, and this quickly becomes infeasible.

3.6 Summary

In this chapter a CCS model of Solid State Interlocking has been developed to focus on properties of Geographic Data, particularly the route locking data in the *PRR*, *PFM*, and *FOP* data files. The basic model in Section 3.2 was defined by translating the Geographic Data into CCS—this approach fixes the semantics of the language (or, equivalently, the behaviour attributed to the SSI generic software) by the translation mechanism. The execution model taken selects a single rule for evaluation in each minor cycle. This differs from the Interlocking's usual mode of operation *vis-à-vis* sub-route release in that there are normally many more sub-route release rules than minor cycles—thus several must be executed consecutively as a block within a single cycle. Although the sub-route release data are processed in strict rotation over each SSI major cycle, it is in general difficult to predict which sub-route release rules will be executed in a particular minor cycle. We are therefore obliged to show that safety is preserved by the execution of each sub-route release rule, and not each block.

This feature of the model can also be justified by observing that when compiling the Geographic Data, the signalling engineer has considerable freedom in specifying the order in which these data are listed (and hence executed). Safety, therefore, *should* be independent of the execution order. Not only is this the case for the *FOP* data, but also the *IPT* and *OPT* data which have not been explained in detail. Since railway signalling engineering is as well a matter of maximising the capacity of the network, there will inevitably be many constraints that select a preferred order in addressing the track-side functional modules. Our thesis, however, is that such considerations should never compromise safety.

In any event, the SSI model is at least as general as the foregoing discussion suggests it needs to be since the execution order has been disregarded entirely: this is not

to say that the order is unimportant, merely that nothing has been assumed about it. Clearly one can conceive of a more elaborate model of SSI, and capture much more of the system's behaviour described in Chapter 1, but one has to ask what is to be the purpose served by the model? For this thesis the answer is that it serves the purpose of checking safety properties *of the data*, and only that. The success of the model therefore has to be judged by the validity of the supposition that properties of the data are independent of SSI. Naturally, one must supply an operational interpretation (semantics) for the Geographic Data Language before questions can be formulated about the properties of 'programs' written in it, but this is precisely what the translation to CCS achieves. Later, in Chapter 5, the Geographic Data will be interpreted in a different way, by means of an embedding in higher-order logic, so the question of validity arises again there.

In this chapter, however, the main assumption about safety properties of the data is that they may be expressed as invariants of the internal state of the Interlocking. The safety properties are expressed in the 'state formula' **F** of Section 3.3; $\mathbf{\Phi} = \nu Z.([\mathsf{ctrl}]ff \vee \mathbf{F}) \wedge [-\mathsf{ctrl}]Z$ specifies that it should be invariant. Other ways of expressing this same property will emerge in subsequent chapters, but it is worth noting that the change in Model #1 which introduced the obs_D tags in Image is not strictly necessary if one's purpose is (only) to check properties such as **F**. Recall that this embellishment was mandated by the $\backslash L$ restriction in defining $(\mathsf{Control} \mid \mathsf{Image})\backslash L$, where $L = \mathcal{L}(\mathsf{Image})$. This is the natural way to present the model since the intention, clearly, is that only these two agents communicate via the L actions. But we can easily do without the restriction: the difference is that $(\mathsf{Control} \mid \mathsf{Image})$ can in principle communicate with external agents through get_D and $\overline{\mathsf{put}}_D$ (or their inverses), but this introduces many bogus states to the model. However, we can instead ask the question

$$(\mathsf{Control} \mid \mathsf{Image}) \quad \models \quad \nu Z.(\langle \mathsf{ctrl} \rangle tt \Rightarrow \mathbf{F}) \wedge [-K]Z$$

where $K = \{\mathsf{ctrl}\} \cup \{\mathsf{get}_D, \overline{\mathsf{get}}_D, \mathsf{put}_D, \overline{\mathsf{put}}_D \mid D \in \mathcal{D}\}$. The $[-K]$ modality instructs the local model checker to ignore precisely the (bogus) transitions that arise from the autonomous movements of these two agents. The above is thus equivalent to

$$(\mathsf{Control} \mid \mathsf{Image})\backslash L \quad \models \quad \nu Z.(\langle \mathsf{ctrl} \rangle tt \Rightarrow \mathbf{F}) \wedge [-\mathsf{ctrl}]Z$$

with the choice of presentation being governed mainly by the efficiency of the model checker's representation of the model. Notice that in the reformulation (without the $\backslash L$) the obs_D tags are no longer needed since **F** can then be expressed in terms of the $\overline{\mathsf{get}}_D$ actions of Image, and these do not cause $(\mathsf{Control} \mid \mathsf{Image})$ to change state.

While safety properties may generally be expressed as state predicates like **MX**, **PT** and **RT**, it is clear that not all properties of the Geographic Data are independent

of the execution order imposed. Temporal properties expressing eventualities may be a case in point: if a route is cancelled the sub-routes will eventually be free. Well this may in fact be independent of the execution order, but in the current model a fairness assumption (to ensure that all *FOP* rules are executed, and that trains progress) is needed for the proof. Moreover, if a time limit were to be specified then not only does order matter, but a notion of major or minor cycle would be needed in the model to provide a time reference. Presently the model lacks any notion of time—principally because the data cannot express anything about clocks or cycles. Later, in Chapter 6, we shall need to introduce a weak notion of time when looking at safety properties of the inter-SSI communications protocol.

For the moment though we have enough difficulties dealing with the invariance proof we have assigned ourselves. Using the model checking facilities of the Concurrency Workbench we were able to prove that WEST satisfies **F** invariantly, and may therefore conclude that the Geographic Data for this scheme are safe (with respect to the properties encoded in **F**). This result was made possible by appealing to the jumping trains axiom and the agent transformations described in Section 3.4 which were highly effective in controlling the state space of the model. However, it is clear by the evidence of the programming experiments described in Section 3.5 that this particular approach to mechanical verification will not scale beyond the simple example: the number of reachable safe states is too great for direct analysis. For this reason global model checking is not explored further as a means to formally verify safety properties of Geographic Data. However, in the next chapter we shall briefly exercise the modal μ-calculus local model checker on the present model: the inductive nature of the algorithm reveals a simple proof method which relieves us of the need to represent the state space of the model at all.

Chapter 4

Proving Safety Properties of Geographic Data

With the basic model of SSI having been explained in the previous chapter, the focus here will be the invariance *proof*. Section 4.2 begins with a sketch of the proof tableau constructed by the local model checking algorithm for the modal μ-calculus because this reveals a simple inductive proof strategy. This leads, in Section 4.3, to a discussion of the fundamental concept underlying our technical definition of safety: the mathematical principle of co-induction. Reinterpreting the model in this light in Section 4.4, we then demonstrate some of the individual proof steps needed to verify that the data are safe. In Section 4.5 we shall look at a number of methods by which these mathematical arguments can be presented within the constraints of a formal logic—in anticipation of Chapter 5 where a proof tool is devised to automate the safety analysis.

4.1 Introduction

Model #2 and the invariant Φ have been designed with the intention of providing a fully automatic proof. In principle, all that is required to achieve full automation in a proposed 'data checker' is to translate the Geographic Data into a formal language such as CCS, express the safety properties in an expressive logic like the modal μ-calculus, and pass the problem on to a sufficiently powerful model checker. In this enterprise the difficulties that emerge are due to the huge size of the state spaces involved.

Yet the problem is not with the model *per se*: while the representation can be changed the model cannot readily be made 'more abstract' since a bit is a bit, and every bit in the image of the railway is significant—at least, all those considered in the *PRR* and *FOP* data are significant. Nor does the problem lie with the safety property: that a state satisfies the formula **F** is easy to establish, and all the safety critical states of the machine must satisfy this property however it is expressed. In fact, the problem lies with the *proof*, and here there is much scope for better abstraction.

The starting point in Section 4.2 is to consider the tableau proof method for the modal μ-calculus due to Stirling and Walker [91]. This is not mere diversion because the structure of the proof tree that the algorithm constructs reveals a remarkable regularity which derives from the structure of the logical formula as much as it does from the structure of the model. It turns out that the enormous complexity of the full proof tree can be folded into a rather small *partial tableau*: then a simple induction argument leads to a highly efficient proof strategy which is linear in the number of rules in the Geographic Data, and independent of the number of states of the model. The proof that the Geographic Data are correct with respect to the formula **F** is therefore reduced to the problem of showing that the individual inductive steps are sound. That is to say, we transfer the problem from that of showing that the model's states are safe, to that of showing that the transitions *preserve safety*. This gives a much more direct proof that the Geographic Data are safe.

The mathematical principle which underlies the invariance proof is that of co-induction. In Section 4.3 we review some of the theory involved, and show how our SSI model and the proof of safety are related to this notion. The model does not survive this analysis completely unscathed however: in setting up the proof as a proof by co-induction it is much more convenient to discard the Image component entirely, representing it instead by a variable (a state vector S) on which the Control operates. This does not change the model's semantics, only its presentation. However, this change does have an impact on the particular formula chosen to express the safety properties identified in the previous chapter.

Section 4.4 draws out some of the details of the proofs needed to show that the *FOP* and *PRR* data are safe. It must be said that these details do not make very interesting reading, but they are important to consider if we are to recover a fully mechanised proof procedure. The proof schemas worked out here are the basis of the *tactics* developed in Section 5.4, and expose a technical difficulty which suggests that the safety property defined in Section 3.3 is too weak. Even if it were not too weak, it turns out we should have difficulty proving the *PRR* data correct with respect to that particular formulation of the property since the tests in these rules do not instantiate enough of the key variables. The difficulty is in proving the mutual exclusion property for track circuits on the route where no sub-route over them is tested in the precondition of the rule: we need to somehow *infer* the status of the intermediate sub-routes from the status of those that are tested in the rule. This problem arises because of the abstraction introduced here in representing the image of the railway, a problem that does not appear when model checking since then all states and state variables are properly instantiated.

Precisely how to strengthen the invariant will not be discussed until Chapter 5 where these proof ideas are formalised and implemented in the HOL theorem prover.

Chronologically, the details of Section 4.4 and the difficulty of proving the invariant in the absence of complete information about the configuration of the internal state predate the other material in this chapter [74]. The 'correct' formulation of **RT**, the component of **F** which needs to be strengthened, only became apparent when the attempt was made to formalise the proof steps in the theorem prover. Moreover, it was only with later analysis that the precise rôle co-induction plays in these arguments became clear. Thus in presenting here a coherent logical progression of these ideas concerning the safety analysis of Geographic Data, it has been necessary to conceal their chronological progression. Such deception is sometimes essential in the interests of making a clear presentation!

In Section 4.5 we round off the discussion of the invariance proof by demonstrating that one can express the same model and invariant in a variety of formalisms, arriving at a similar proof obligation in each case. In TLA the problem is tackled from a purely logical standpoint, reducing the invariance proof to (what amounts to) Boolean satisfiability. In UNITY one can capture the SSI model in a nondeterministic program reminiscent of that of Model #1, and the proof turns out to be the same as that illustrated in Section 4.4. Thirdly, we set up the proof in the style of Floyd-Hoare logic, and show that the same *verification conditions* arise here too. This convergence of methods suggests that the choice of an appropriate environment in which to mechanise the invariance will be largely governed by efficiency considerations.

4.2 Tableau Proofs in Local Model Checking

When modelled at the abstract level of the previous chapter, the Leamington Spa signalling scheme mentioned by Cribbens [24] has between 2^{48} and 2^{230} states ($\mathcal{T} = 48$, $\mathcal{U} + \mathcal{R} + \mathcal{P} = 96 + 71 + 15$). *With* the jumping trains abstraction of Section 3.4.1 one can guesstimate the reachable state space of the model by placing three to four copies of THORNTON JN. in parallel (circa 10^{21} states). Dealing with such inherent computational complexity is the principal difficulty in formally verifying properties of Geographic Data through model checking. It is nevertheless instructive to unfold a small portion of the tableau that the local model checking algorithm constructs since this illustrates how the verification conditions that need to be established are obtained. This leads to a simple inductive method for proving the invariant.

4.2.1 Unfolding Proof Tableaux

We proceed here with the tagged Model #2 and adhere to the notation used by Stirling and Walker throughout in unfolding the proof tree [90, and Appendix B.2 for details of the algorithm]. The numbered paragraphs below correspond to the numbered

nodes in the displayed proof tree. Let I_0 stand for the initial state of Image, and let $\Phi = \nu Z.([\mathsf{ctrl}]ff \vee \mathbf{F}) \wedge [-\mathsf{ctrl}]Z$. Since the algorithm is sound and complete, $(\mathsf{Control} \mid I_0)\backslash L \models \Phi$ if and only if there exists a successful tableau with root sequent $(\mathsf{Control} \mid I_0)\backslash L \vdash \Phi$. The computation begins:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{1}\ (\mathsf{Control} \mid I_0)\backslash L \vdash \nu Z.([\mathsf{ctrl}]ff \vee \mathbf{F}) \wedge [-\mathsf{ctrl}]Z}{(\mathsf{Control} \mid I_0)\backslash L \vdash U}}{(\mathsf{Control} \mid I_0)\backslash L \vdash ([\mathsf{ctrl}]ff \vee \mathbf{F}) \wedge [-\mathsf{ctrl}]U}}{\dfrac{\mathbf{2}\ (\mathsf{Control} \mid I_0)\backslash L \vdash [\mathsf{ctrl}]ff \vee \mathbf{F}}{(\mathsf{Control} \mid I_0)\backslash L \vdash \mathbf{F}} \qquad \dfrac{\mathbf{3}\ (\mathsf{Control} \mid I_0)\backslash L \vdash [-\mathsf{ctrl}]U}{\iddots \quad \vdots \quad \ddots}}}$$

1 The first proof rule introduces a constant $U \stackrel{\text{def}}{=} \nu Z.([\mathsf{ctrl}]ff \vee \mathbf{F}) \wedge [-\mathsf{ctrl}]Z$. This constant is immediately unrolled, the rule replacing all (free) occurrences of Z in the formula by U, and the tree splits due to a conjunction. Both branches must be successful if a successful tableau is to be found for the root sequent. The side condition to the rule for unrolling the constant is that no node in the tree above should be labelled with the same sequent; if there is such a node the constant is not unrolled, and the node is declared a terminal.

2 In general, disjunctive goals introduce an element of choice in the proof tree; but here, since $(\mathsf{Control} \mid I_0)\backslash L \xrightarrow{\mathsf{ctrl}}$, the left-hand disjunct inevitably leads to an unsuccessful leaf. Essentially, the obligation at this point is to show that $I_0 \models \mathbf{F}$. That I_0 does or does not satisfy $\mathbf{F}$ is easy to ascertain since it is only necessary to examine the immediate capabilities of this agent; if $I_0 \not\models \mathbf{F}$ we might as well halt the algorithm immediately since no successful tableau can exist.

3 There is considerable fanout here since the proof rule for the box modality insists that every derivative of $(\mathsf{Control} \mid I_0)\backslash L$ must be considered, other than the 'deadlock' reached via $\xrightarrow{\mathsf{ctrl}}$. Next actions due to tags in the Image component lead immediately to a node labelled with the same sequent as **1**. Such nodes are successful terminals and need not be considered further. However, the algorithm must also consider the next actions of the Control, and there is one branch corresponding to each panel route request and sub-route release rule in the data (as well as those from the Input and Cancel components). Just one derivative is considered here, corresponding to the sub-route release rule T_1^{ca} `f if` T_1 `c` , R_{02} `xs \ ..`

$$\begin{aligned}
\mathsf{U1CA} &= \mathsf{get}_{T_1}(\mathtt{c}).\mathsf{U1CA}_1 + \mathsf{get}_{T_1}(\mathtt{o}).\mathsf{Control} \\
\mathsf{U1CA}_1 &= \mathsf{get}_{R_{02}}(\mathtt{xs}).\mathsf{U1CA}_2 + \mathsf{get}_{R_{02}}(\mathtt{s}).\mathsf{Control} \\
\mathsf{U1CA}_2 &= \overline{\mathsf{put}}_{T_1^{ca}}(\mathtt{f}).\mathsf{Control}
\end{aligned}$$

Without loss of generality we may suppose $I_0 \models \langle \mathsf{obs}_{T_1}(\mathtt{c}) \rangle tt$; otherwise this branch in the proof tree leads immediately to a node labelled with the sequent **1**.

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{3}\ (\mathsf{Control} \mid I_0) \backslash L \vdash [-\mathsf{ctrl}]U}{(\mathsf{U1CA}_1 \mid I_0) \backslash L \vdash U}}{(\mathsf{U1CA}_1 \mid I_0) \backslash L \vdash (\langle \mathsf{ctrl} \rangle ff \vee \mathbf{F}) \wedge [-\mathsf{ctrl}]U}}{\dfrac{\mathbf{4}\ (\mathsf{U1CA}_1 \mid I_0) \backslash L \vdash \langle \mathsf{ctrl} \rangle ff \vee \mathbf{F}}{(\mathsf{U1CA}_1 \mid I_0) \backslash L \vdash \langle \mathsf{ctrl} \rangle ff} \quad \dfrac{\mathbf{5}\ (\mathsf{U1CA}_1 \mid I_0) \backslash L \vdash [-\mathsf{ctrl}]U}{\vdots}}}$$

4,6 The situation here is similar to that in **2** except that both the possible subtrees are successful (assuming $I_0 \models \mathbf{F}$). One can choose the branch in the proof that involves the least computation—in this case the left-hand subtree since the agent cannot immediately perform a ctrl action. This terminal is successful.

5 As with **3**, any next action due to an autonomous move in Image leads directly to a successful terminal (labelled with the same sequent as **3**). Otherwise there are fewer subtrees to examine as $\mathsf{U1CA}_1$ unfolds:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{5}\ (\mathsf{get}_{R_{02}}(\mathtt{xs}).\mathsf{U1CA}_2 + \mathsf{get}_{R_{02}}(\mathtt{s}).\mathsf{Control} \mid I_0) \backslash L \vdash [-\mathsf{ctrl}]U}{(\mathsf{U1CA}_2 \mid I_0) \backslash L \vdash U}}{(\mathsf{U1CA}_2 \mid I_0) \backslash L \vdash (\langle \mathsf{ctrl} \rangle ff \vee \mathbf{F}) \wedge [-\mathsf{ctrl}]U}}{\dfrac{\mathbf{6}\ (\mathsf{U1CA}_2 \mid I_0) \backslash L \vdash \langle \mathsf{ctrl} \rangle ff \vee \mathbf{F}}{(\mathsf{U1CA}_2 \mid I_0) \backslash L \vdash \langle \mathsf{ctrl} \rangle ff} \quad \dfrac{\mathbf{7}\ (\mathsf{U1CA}_2 \mid I_0) \backslash L \vdash [-\mathsf{ctrl}]U}{(\mathsf{Control} \mid I_1) \backslash L \vdash U}}}$$

This derivation assumes $I_0 \models \langle \mathsf{obs}_{R_{02}}(\mathtt{xs}) \rangle tt$ since, for the other case, the node labelled $(\mathsf{Control} \mid I_0) \backslash L \vdash U$ is a successful terminal as it is appears also at **1**.

7 Here the situation is analogous to **1** except now the Image has (or at least may have) changed in respect of the variable T_1^{ca}. If I_0 and I_1 are identical this is a successful terminal, and this branch of the proof tree (from **3**) terminates with all its leaves successful according to the algorithm's termination criteria. On the other hand, if these two agents are different, the tableau continues to unfold:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathbf{7}\ \overline{\mathsf{put}}_{T_1^{ca}}(\mathtt{f}).\mathsf{Control} \vdash [-\mathsf{ctrl}]U}{(\mathsf{Control} \mid I_1) \backslash L \vdash U}}{(\mathsf{Control} \mid I_1) \backslash L \vdash (\langle \mathsf{ctrl} \rangle ff \vee \mathbf{F}) \wedge [-\mathsf{ctrl}]U}}{\dfrac{\mathbf{8}\ (\mathsf{Control} \mid I_1) \backslash L \vdash \langle \mathsf{ctrl} \rangle ff \vee \mathbf{F}}{(\mathsf{Control} \mid I_1) \backslash L \vdash \mathbf{F}} \quad \dfrac{(\mathsf{Control} \mid I_1) \backslash L \vdash [-\mathsf{ctrl}]U}{\iddots \quad \vdots \quad \ddots}}}$$

In order to make the model checking slightly more efficient we can exploit *confluence* in the tableau: that is, identifying nodes under the right-hand subtree from **7**, and nodes labelled with the same sequents under the other branches from **3**. This optimisation does not affect the soundness of the algorithm [19].

8 The situation here is similar to that at **2**: this is a control state, so the safety property **F** must be verified. Again, this is an easy check whether $I_1 \models \mathbf{F}$, where I_1 corresponds to I_0, except in respect of the variable T_1^{ca}.

There is no need to proceed further with the tableau. It is clear, intuitively at least, that the algorithm sketched above explores all states of the model. This arises from the box modality, $[-\mathsf{ctrl}]$, which specifies that all states, other than those reached via the ctrl action, must satisfy the invariant—the model checker duly examines them all. As indicated in Section 3.4.3, this is not a weakness particular to the local model checking algorithm: it is a general problem where such a simple invariant entails what amounts to an exhaustive search of all the system's states. Neither Cleaveland's more efficient algorithm [19], nor the iterative method of Clark *et al.* [18], would fare any better here.

Nevertheless, the structure of the proof tree sketched above reveals a simple induction principle: if I_0 is an arbitrary initial state and $I_0 \models \mathbf{F}$ *implies* $I_1 \models \mathbf{F}$, then whenever I_0 is instantiated to a particular (initial) state we can deduce that the successor $I_1 \models \mathbf{F}$ when I_0 does so. The proposition asserts that the sub-route release rule for T_1^{ca} (say) preserves the invariance of **F**, and this can be used to prune the proof tree. We now make this idea more precise.

4.2.2 Partial Tableaux

Definition 4.1 Let $\Phi \stackrel{\text{def}}{=} \nu Z.([\mathsf{ctrl}]ff \vee \mathbf{F}) \wedge [-\mathsf{ctrl}]Z$, and define $(\mathsf{Control} \mid \mathsf{Image})\backslash L$ as in Model #2. Let I represent an arbitrary configuration of Image. A **partial tableau** is a proof tree for $(\mathsf{Control} \mid I)\backslash L \vdash \Phi$ whose terminals are labelled by one of

- $(\mathsf{Control} \mid I)\backslash L \vdash \mathbf{F}$
- $(\mathsf{Control} \mid I_j)\backslash L \vdash U$, for some I_j and where $U \stackrel{\text{def}}{=} \Phi$
- $(\mathsf{Control}' \mid I)\backslash L \vdash [\mathsf{ctrl}]ff$, for some $\mathsf{Control}' \not\equiv \mathsf{Control}$

and where there is exactly one node $(\mathsf{Control} \mid I')\backslash L \vdash U$, for some I', on the branch from the root sequent to any terminal labelled $(\mathsf{Control} \mid I_j)\backslash L \vdash U$. □

Call $I_j \not\equiv I$ a *one-step successor* of the initial state I if $(\mathsf{Control} \mid I_j)\backslash L \vdash U$ is such a teminal in the partial tableau $\boldsymbol{T}$. Due to the shape of the invariant Φ, the partial tableau for $(\mathsf{Control} \mid I)\backslash L \vdash \Phi$ has the overall structure shown in Figure 4.1. A partial tableau is constructed by the proof rules for the local model checking algorithm of Stirling and Walker, but since the initial state I is uninstantiated we observe that whenever the box rule is applied to nodes like $(\mathsf{Control}' \mid I')\backslash L \vdash [-\mathsf{ctrl}]U$ this will cause maximal fanout in the tableau since, in particular, all next transitions of $\mathsf{Control}'$

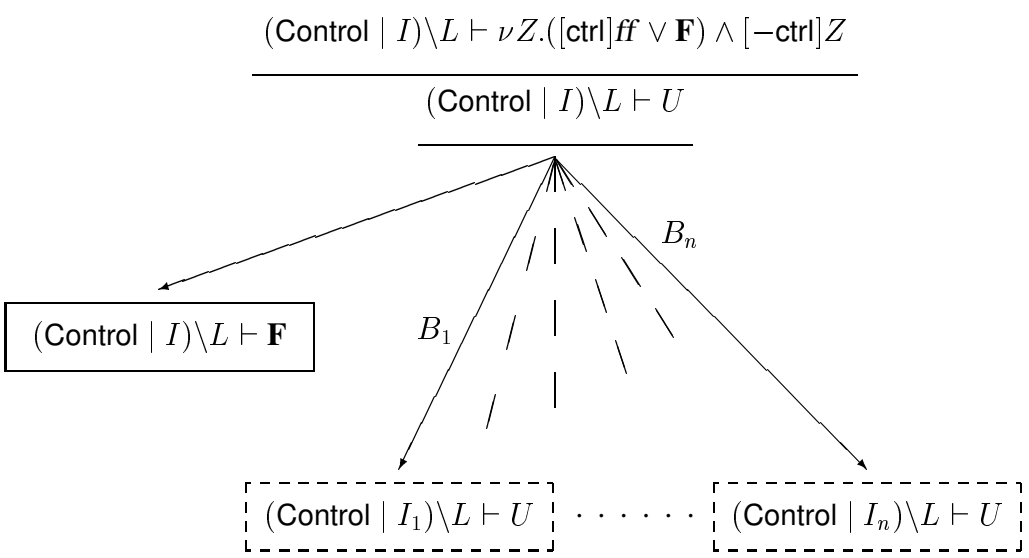


Figure 4.1: A Partial tableau

must be recorded. Clearly this gives rise to n branches from $(\mathsf{Control} \mid I)\backslash L \vdash U$ that terminate with nodes of the form $(\mathsf{Control} \mid I_j)\backslash L \vdash U$, for $I_j \not\equiv I$, where n is the fanout of the Control component of the model as defined in Section 3.2.2.

Now given a partial tableau $\boldsymbol{T}$ we can associate a *branch predicate* B_j with each branch that leads to a terminal of the form $(\mathsf{Control} \mid I_j)\backslash L \vdash U$. When $I_j \equiv I$ we let $B_j = \mathit{tt}$, otherwise B_j is deduced by observing the actions performed (by the Image component) in arriving at the terminal $(\mathsf{Control} \mid I_j)\backslash L \vdash U$.

Definition 4.2 Given a partial tableau $\boldsymbol{T}$, a **branch predicate** B_j has the following syntax, where $\overline{\mathsf{get}}_{D_i}(v_i), \mathsf{put}_{D_i}(v_i) \in \mathcal{L}(\mathsf{Image})$:

$$\begin{aligned} B_j &\stackrel{\text{def}}{=} (\mathbf{F} \wedge G_j) \Rightarrow P_j(\mathbf{F}) \\ G_j &\stackrel{\text{def}}{=} \bigwedge_i \langle \overline{\mathsf{get}}_{D_i}(v_i) \rangle \mathit{tt}, \quad i \geq 0 \\ P_j(\Phi) &\stackrel{\text{def}}{=} [\mathsf{put}_{D_1}(v_1)][\cdots][\mathsf{put}_{D_n}(v_n)]\Phi, \quad n \geq 1 \end{aligned}$$

Note that $G_j = \mathit{tt}$ when $i = 0$. Let $\mathcal{B}_{\boldsymbol{T}}$ be the set of branch predicates associated with $\boldsymbol{T}$. A partial tableau is **exhaustive** iff $I \models B_j$ for all $B_j \in \mathcal{B}_{\boldsymbol{T}}$. □

Lemma 4.1 Giver a partial tableau $\boldsymbol{T}$, and $B_j \in \mathcal{B}_{\boldsymbol{T}}$:

1. $(\mathsf{Control} \mid I)\backslash L \vdash \mathbf{F}$ iff $I \models \mathbf{F}$
2. $I \models B_j$ iff $I \models \mathbf{F} \wedge G_j$ implies $I_j \models \mathbf{F}$

for $(\mathsf{Control} \mid I_j)\backslash L \vdash U$ a terminal in $\boldsymbol{T}$. □

Proof The first part follows directly from the definition of **F**, and from the proof rules for the local model checking algorithm. For the second part we use Definition 4.2 and the definition of satisfaction for Hennessey-Milner logic. For $B_j \in \mathcal{B}_{\mathcal{T}}$:

$$\begin{array}{ll} & I \models B_j \\ \text{iff} & I \models (\mathbf{F} \wedge G_j) \Rightarrow P_j(\mathbf{F}) \\ \text{iff} & I \models (\mathbf{F} \wedge G_j) \text{ implies } I \models P_j(\mathbf{F}) \end{array}$$

Since $\mathsf{Reg}_D(u) \xrightarrow{\mathsf{put}_D(v)} \mathsf{Reg}_D(v)$ in any state holding u, it follows that $I \models P_j(\mathbf{F})$ if and only if $I_j \models \mathbf{F}$. ■

Proposition 4.2 If there exists an exhaustive partial tableau for $(\mathsf{Control} \mid I)\backslash L \vdash \mathbf{\Phi}$, and if $I_0 \models \mathbf{F}$ for some initial configuration of Image, then $(\mathsf{Control} \mid I_0)\backslash L \models \mathbf{\Phi}$. □

Proof $(\mathsf{Control} \mid I_0)\backslash L \models \mathbf{\Phi}$ if and only if we can construct a successful tableau for $(\mathsf{Control} \mid I_0)\backslash L \vdash \mathbf{\Phi}$ using the proof rules for the local model checking algorithm. We show that this is always possible if there exists an exhaustive partial tableau for $(\mathsf{Control} \mid I)\backslash L \vdash \mathbf{\Phi}$, as long as $I_0 \models \mathbf{F}$.

Consider $(\mathsf{Control} \mid I_0)\backslash L \vdash \mathbf{\Phi}$, and unfold the proof tree for this goal in a breadth first manner until all one-step successors of I_0 have been encountered. Terminals of this (partially constructed) proof tree will have one of the forms:

(1) $(\mathsf{Control}' \mid I_0)\backslash L \vdash [\mathsf{ctrl}]ff$ for $\mathsf{Control}' \not\equiv \mathsf{Control}$, or

(2) $(\mathsf{Control} \mid I_0)\backslash L \vdash \mathbf{F}$, or

(3) $(\mathsf{Control} \mid I_k^1)\backslash L \vdash U$

where I_k^1 is a one-step successor of I_0. The only other possibility, namely nodes like $(\mathsf{Control}' \mid I')\backslash L \vdash \mathbf{F}$ for some I' and $\mathsf{Control}' \not\equiv \mathsf{Control}$, can be discounted since we need only show that there exists a successful tableau for the root sequent.

Nodes such as (1) are successful terminals by the local model checking criteria since the agent cannot immediately perform a $\xrightarrow{\mathsf{ctrl}}$ action. Nodes such as (2) are the roots of successful subtrees by Lemma 4.1.1, since $I_0 \models \mathbf{F}$ by hypothesis. For nodes such as (3) there are two cases to consider. If $I_k^1 \equiv I_0$ then this is a successful terminal since then node $(\mathsf{Control} \mid I_0)\backslash L \vdash U$ already appears at a higher level on the same branch in the proof tree under construction. Otherwise $I_k^1 \not\equiv I_0$ and so we must continue to unfold the proof tree from all such nodes since these are *not* terminals by the local model checking criteria.

Take some such $(\mathsf{Control} \mid I_k^1)\backslash L \vdash U$, and unfold the proof tree in a breadth first manner until all one-step successors of I_k^1 have been encountered. Terminals in this subtree will have one of the forms:

(4) $(\mathsf{Control}' \mid I_k^1)\backslash L \vdash [\mathsf{ctrl}]ff$ for $\mathsf{Control}' \not\equiv \mathsf{Control}$, or

(5) $(\mathsf{Control} \mid I_k^1)\backslash L \vdash \mathbf{F}$, or

(6) $(\mathsf{Control} \mid I_k^2)\backslash L \vdash U$

where I_k^2 is a one-step successor of I_k^1. Nodes such as (5) are successful terminals for the same reason as (1) above. To see that (4) is the root of a successful subtree it is enough, by Lemma 4.1.1, to show that $I_k^1 \models \mathbf{F}$. Given that I_k^1 is a one-step successor of I_0 we can find B_k such that $B_k = (\mathbf{F} \wedge G_k) \Rightarrow P_k(\mathbf{F})$, and $I_0 \models G_k$. Now suppose

$$\neg(I_0 \models \mathbf{F} \wedge G_k \text{ implies } I_0 \models P_k(\mathbf{F})) \qquad (\dagger)$$

(that is to say $I_0 \not\models B_k$, using Lemma 4.1.2). Well, since there exists an exhaustive partial tableau for $(\mathsf{Control} \mid I)\backslash L \vdash \mathbf{\Phi}$ then, in particular, $I \models B_k$ with $B_k \in \mathcal{B}_{\boldsymbol{T}}$, and this contradicts (†) above. Thus $I_0 \models \mathbf{F} \wedge G_k$ implies $I_0 \models P_k(\mathbf{F})$; since $I_0 \models \mathbf{F}$ it follows that $I_0 \models \mathbf{F} \wedge G_k$ and, *modus ponens*, $I_0 \models P_k(\mathbf{F})$. The conclusion $I_k^1 \models \mathbf{F}$ therefore follows by the definition of Image.

Before concluding that $(\mathsf{Control} \mid I_k^1)\backslash L \vdash U$ is the root of a successful subtree we have to show that nodes such as (6) also have only successful terminals according to the local model checking criteria. But given that the model $(\mathsf{Control} \mid I)\backslash L$ is finite state we can repeat the forgoing argument (with I_k^1 playing the rôle of I_0, *etc.*) along every branch of the tree, and so conclude that all successors I_k^n of I_0, for some finite n, satisfy the safety property $\mathbf{F}$. Thus $(\mathsf{Control} \mid I_0)\backslash L \vdash \mathbf{\Phi}$, and soundness of the local model checking algorithm guarantees $(\mathsf{Control} \mid I_0)\backslash L \models \mathbf{\Phi}$ as required. ■

Proposition 4.2 is useful since it indicates that we do not have to enumerate states of the SSI in order to prove the invariant. It is enough, though perhaps still not easy, to establish that if an arbitrary initial state satisfies the invariant, so do its immediate successors. Then, given any initial state (or more generally, any set of initial states), if the initial state satisfies the safety property this will be maintained through all future evolution of the system because the $\mathsf{Control}$ permits only safe transitions. This is not to insist that the initial state of the SSI satisfies our safety property, only that the 'initial state' represent some reasonable state for the machine to be in.

The drawback, in using this result, is that in order to find the required lemmas we have to go outside the proof system of the modal μ-calculus—though this does not invalidate the mathematical argument of course.

4.3 Invariance & Co-induction

Safety properties are associated with greatest fixed points in the modal μ-calculus. This was demonstrated in Larsen's work on modal equations with recursion [54, 89].

An example, perhaps the simplest, is the equation $Z \stackrel{\text{def}}{=} \Phi \wedge [K]Z$ where Z does not appear in Φ. Larsen defines satisfaction, $E \models \Psi$, indirectly in terms of the (inductively defined) semantics of Ψ—given as $\|\Psi\|$, the set of states in the model that have the property. Larsen shows that the $\|\cdot\|$ operator is monotonic, so maximal and minimal solutions exist for such recursive modal equations. In particular, the greatest fixed point of a function $\mathcal{F} : \mathcal{P} \to \mathcal{P}$ is the union of all pre-fixed points

$$\text{gfp}\,\mathcal{F} = \bigcup \{\mathcal{E} \subseteq \mathcal{P} \mid \mathcal{E} \subseteq \mathcal{F}(\mathcal{E})\} \qquad (*)$$

where $\mathcal{P}$, here, is the universal set of (CCS) processes, or states. (The set $\mathcal{E}$ is a *pre-fixed point* of $\mathcal{F}$ iff $\mathcal{E} \subseteq \mathcal{F}(\mathcal{E})$.) Stirling [89] has extended these ideas to define the semantics of the modal μ-calculus in which least and greatest fixed points may become arbitrarily entangled (the pertinent details are sketched in Appendix B.2). But in our simple example where Z does not appear in Φ, $\mathcal{F}$ is the function

$$\begin{aligned}
\mathcal{F}(\mathcal{E}) &= \|\Phi \wedge [K]Z\| \\
&= \|\Phi\| \cap \|\,[K]\,\|\mathcal{E} \\
&= \|\Phi\| \cap \{E \in \mathcal{P} \mid \textit{if } E \xrightarrow{a} E' \textit{ and } a \in K\textit{, then } E' \in \mathcal{E}\}
\end{aligned}$$

which is easily seen to be monotonic because of the intersection with $\|\Phi\|$ which is a fixed set (of states). $\|\Phi\|$ is interpreted as the set of states that satisfy Φ; the other component is a set of states E such that if E performs an action $a \in K$, the derivative E' is contained in $\mathcal{E}$.

Now the greatest fixed point of $\mathcal{F}$ is written $\nu Z.\Phi \wedge [K]Z$ in the modal μ-calculus, and the right-hand side of $(*)$ gives the meaning of this temporal formula:

$$\bigcup \{\mathcal{E} \subseteq \mathcal{P} \mid \mathcal{E} \subseteq \|\Phi\| \cap \{E \in \mathcal{P} \mid \textit{if } E \xrightarrow{a} E' \textit{ and } a \in K\textit{, then } E' \in \mathcal{E}\}\}$$

That is to say, it is the union of the sets $\mathcal{E} \subseteq \mathcal{P}$ such that $\mathcal{E} \subseteq \|\Phi\|$, *and* $\mathcal{E}$ is a subset of $\{E \in \mathcal{P} \mid \textit{if } E \xrightarrow{a} E' \textit{ and } a \in K\textit{, then } E' \in \mathcal{E}\}$. To establish that some set of states Q is a pre-fixed point of $\mathcal{F}$ we show $Q \subseteq \mathcal{F}(Q)$. This is the case if one can prove that $E \in Q$ implies $E \in \mathcal{F}(Q)$—in other words we prove $E \in Q$ implies

- $E \models \Phi$, and
- whenever $E \xrightarrow{a} E'$ with $a \in K$, $E' \in Q$.

This is an instance of *co-induction*.

In a motivating article [65] Milner and Tofte exemplify the use of co-induction as a device for reasoning about non-well-founded sets. They prove the consistency between the static and the dynamic semantics of a simple functional language with recursion, co-induction being used to show that recursive functions are well typed. Paraphrasing [65], the principle may be stated thus:

> *If U is any set, $\mathcal{F} : 2^U \to 2^U$ is a monotonic function, and R is the greatest fixed point of $\mathcal{F}$, then $Q \subseteq \mathcal{F}(Q)$ implies $Q \subseteq R$ for any $Q \subseteq U$.*

Taking U to be the set of SSI images of the railway, we wish to show that $\{I \in U \mid I \models \mathbf{F}\}$ is contained in the greatest fixed point of a certain monotonic operator ($\mathcal{C}$, say) which is induced by our encoding of the Geographic Data. This is the *meaning* of the safety property expressed in the temporal formula $\nu Z.([\mathsf{ctrl}]ff \vee \mathbf{F}) \wedge [-\mathsf{ctrl}]Z$. In pursuing this idea we discard the Image component of the CCS model, replacing it instead by an n-tuple, S, of appropriate variables to represent the image of the railway. States of the model are henceforth written $\mathsf{Control}(S)$. We could refer the components of S as $S.T_1$, *etc.*, but generally prefer their usual names in the sequel.

Now, let C, C', C_i range over states of Control, S, S', S_i over 2^n, and define a new transition system, $(\mathcal{M}, \mathcal{L}(\mathsf{Control}), T)$, where $\mathcal{M} = \{C(S) \mid S \in 2^n\}$ and the transition relation T is defined by the clauses:

i. $C(S) \xrightarrow{\mathsf{get}_D(v)} C'(S)$ & $S.D = v$ if and only if $\dfrac{C \xrightarrow{\mathsf{get}_D(v)} C' \quad I \xrightarrow{\overline{\mathsf{get}}_D(v)} I}{(C \mid I)\backslash L \xrightarrow{\tau} (C' \mid I)\backslash L}$

ii. $C(S) \xrightarrow{\overline{\mathsf{put}}_D(v)} C'(S[v/D])$ if and only if $\dfrac{C \xrightarrow{\overline{\mathsf{put}}_D(v)} C' \quad I \xrightarrow{\mathsf{put}_D(v)} I'}{(C \mid I)\backslash L \xrightarrow{\tau} (C' \mid I')\backslash L}$

iii. $C(S) \xrightarrow{a} C'(S)$ if and only if $\dfrac{C \xrightarrow{a} C'}{(C \mid I)\backslash L \xrightarrow{a} (C' \mid I)\backslash L}$

where, in the last case, $a \in \mathcal{L}(\mathsf{Control})$ is any action other than a get_D or a $\overline{\mathsf{put}}_D$ action. $\mathcal{M}$ is much too large, representing as it does the space of all possible SSI configuration rather than just the reachable ones (from some initial set), but our interest here is in the coarser transition relation, $\rightsquigarrow \subseteq \mathcal{M} \times \mathcal{M}$ (implicitly defined in Section 3.5). This is defined as follows: $\mathsf{Control}(S) \rightsquigarrow \mathsf{Control}(S')$ iff there exist $a_0, a_1, \ldots, a_n \in \mathcal{L}(\mathsf{Control})$ and states $C_1(S_1), \ldots, C_n(S_n) \in \mathcal{M}$ such that

$$\mathsf{Control}(S) \xrightarrow{a_0} C_1(S_1) \xrightarrow{a_1} \cdots C_n(S_n) \xrightarrow{a_n} \mathsf{Control}(S')$$

with $\{C_1, \ldots, C_n\} \models [\mathsf{ctrl}]ff$ (*i.e.*, none are control points). The operator $\mathcal{C}$ alluded to earlier is therefore defined by

$$\mathcal{C}(\mathcal{S}) \stackrel{\text{def}}{=} \{S' \mid \exists S \in \mathcal{S}.\mathsf{Control}(S) \rightsquigarrow \mathsf{Control}(S')\}$$

where $\mathcal{S} \subseteq 2^n$.

Finally recall that, as a modal formula, $\mathbf{F}$ was defined in terms of the obs_D tags in Image. In order for the assertion $S \models \mathbf{F}$ to be meaningful in the new setting $\mathbf{F}$ must be redefined accordingly. But this is easy for if S represents Image, in some configuration, then $\mathsf{Image} \models \langle \mathsf{obs}_D(v) \rangle tt$ if and only if $S.D = v$. Returning, therefore,

to the induction principle given earlier, we define $Q = \{S \in 2^n \mid S \models \mathbf{F}\}$. Then to show Q is contained in the greatest fixed point of $\mathcal{C}$ we may prove $Q \subseteq \mathcal{C}(Q)$. It suffices to show instead that $S \models \mathbf{F} \Rightarrow \mathcal{C}(S) \models \mathbf{F}$ (since then the sets are equal). Notice that we do not insist all of Q is reachable from any given initial state in Q; the proof demonstrates only that if the machine is in a state $S \in Q$, it will not leave Q.

4.4 Checking Interlocking Data

There are many ways in which to check $S \models \mathbf{F}$ implies $\mathcal{C}(S) \models \mathbf{F}$ mechanically (*cf.* Section 4.5). The strategies developed below are tailored to the different classes of Geographic Data, and inform the search for proofs in Chapter 5. These sketches illustrate clearly how **MX**, **PT**, and **RT** combine to guarantee safety. We shall find it convenient to use some elementary properties of these invariants—principally that *unlocking* sub-routes does not affect the truth of **MX** or **PT**:

Lemma 4.3 For all sub-routes u, and routes r

1. $S \models \mathbf{MX} \wedge \mathbf{PT} \Rightarrow S[\mathtt{f}/u] \models \mathbf{MX} \wedge \mathbf{PT}$
2. $S \models \mathbf{F} \Rightarrow S[\mathtt{xs}/r] \models \mathbf{F}$

when $\mathbf{F} = \mathbf{MX} \wedge \mathbf{PT} \wedge \mathbf{RT}$. □

Proof This is straightforward given the definitions in Section 3.3.3, and the interpretation that $\mathsf{Image} \models \langle \mathsf{obs}_D(v) \rangle tt$ iff $S.D = v$. It is worth noting that $S \models \mathbf{MX}[a, b]$ iff $S.a = \mathtt{f} \vee S.b = \mathtt{f}$—that is, if and only if $\neg(S.a = \mathtt{l} \wedge S.b = \mathtt{l})$; the second equivalence here is valid because sub-routes *are* propositional variables. ■

4.4.1 Sub-route Release Data

To see how to establish that the invariant holds for the sub-route release rules we shall consider two typical examples, taking the data for T_1^{ca} and T_2^{ba} in WEST. These are the components of the route R_{02} in Figure 2.1. The demonstration only sketches the essential details of the proof as it might proceed by hand: even these steps are laborious, so mechanised support is essential to improve confidence in the results. The uniformity in the sub-route release rules means they can all be verified in much the same way.

Proposition 4.4 The sub-route release rules for T_1^{ca} and T_2^{ba}

$$T_1^{ca}\,\mathtt{f}\ \mathtt{if}\ T_1\,\mathtt{c}, R_{02}\,\mathtt{xs}\ \backslash\ .$$
$$T_2^{ba}\,\mathtt{f}\ \mathtt{if}\ T_2\,\mathtt{c}, T_1^{ca}\,\mathtt{f}\ \backslash\ .$$

preserve the invariance of $\mathbf{F} = \mathbf{MX} \wedge \mathbf{PT} \wedge \mathbf{RT}$. □

Proof Given state S we exhibit the derivation of successor state S' from the definition of the Control. It is required to show that $S \models \mathbf{F}$ implies $S' \models \mathbf{F}$. Lemma 4.3 is used to reduce this to showing $S \models \mathbf{F}$ implies $S' \models \mathbf{RT}$. The values of the variables deduced in the derivation of S' from S are needed to complete the crucial steps.

From the definition of the Control we arrive at the following sequence of transitions leading from a state S to successor states S_1 and S_2:

$$\begin{aligned}
\mathsf{Control}(S) &= \mathsf{get}_{T_1}(\mathtt{c}).(\mathsf{get}_{R_{02}}(\mathtt{xs}).\overline{\mathsf{put}}_{T_1^{ca}}(\mathtt{f}).\mathsf{Control}(S_1) + \\
&\qquad\qquad \mathsf{get}_{R_{02}}(\mathtt{s}).\mathsf{Control}(S)) \\
&+ \mathsf{get}_{T_1}(\mathtt{o}).(\mathsf{get}_{R_{02}}(\mathtt{xs}).\mathsf{Control}(S) + \mathsf{get}_{R_{02}}(\mathtt{s}).\mathsf{Control}(S)) \qquad \text{(i)} \\
\mathsf{Control}(S) &= \mathsf{get}_{T_2}(\mathtt{c}).(\mathsf{get}_{T_1^{ca}}(\mathtt{f}).\overline{\mathsf{put}}_{T_2^{ba}}(\mathtt{f}).\mathsf{Control}(S_2) + \\
&\qquad\qquad \mathsf{get}_{T_1^{ca}}(\mathtt{l}).\mathsf{Control}(S)) \\
&+ \mathsf{get}_{T_2}(\mathtt{o}).(\mathsf{get}_{T_1^{ca}}(\mathtt{f}).\mathsf{Control}(S) + \mathsf{get}_{T_1^{ca}}(\mathtt{l}).\mathsf{Control}(S)) \qquad \text{(ii)}
\end{aligned}$$

Here $S_1 = S[\mathtt{f}/T_1^{ca}]$ and $S_2 = S[\mathtt{f}/T_2^{ba}]$, and the obligation is to show that S_1 and S_2 both satisfy **F** as long as $S \models \mathbf{F}$. It follows from Lemma 4.3 that the successor states satisfy **MX** and **PT** as long as the initial state satisfies **F**. For the **RT** component, inspection of the definition (on page 60) reveals that the updated variables appear in only one conjunct; the others are proven since $S \models F_i \Rightarrow S' \models F_i$ is always true if none of the variables on which S and S' differ appear in F_i. The only term to consider is that defining the invariant for R_{02}:

$$S \models R_{02=\mathtt{s}} \Rightarrow T_1^{ca} = \mathtt{l} \wedge T_2^{ba} = \mathtt{l} \qquad \{\text{since } S \models \mathbf{RT}\}$$

$$S_1 \models R_{02} = \mathtt{xs} \wedge T_1^{ca} = \mathtt{f} \qquad \{\text{from (i) and } S_1 = S[\mathtt{f}/T_1^{ca}]\}$$

The conclusion that $S_1 \models \mathbf{RT}$ follows immediately since the implication is always true when the antecedent is false.

The same conjunct is involved in updating S to S_2, but now there are two cases to consider which depend on whether or not R_{02} is set in S. Firstly:

$$S \models R_{02} = \mathtt{s} \qquad \{\text{by hypothesis}\}$$

$$S \models R_{02=\mathtt{s}} \Rightarrow T_1^{ca} = \mathtt{l} \wedge T_2^{ba} = \mathtt{l} \qquad \{\text{since } S \models \mathbf{RT}\}$$

$$S \models T_1^{ca} = \mathtt{l} \wedge T_2^{ba} = \mathtt{l} \qquad \{\textit{modus ponens}\}$$

but from (ii) it is clear that if the system evolves to S_2 then $S \models T_1^{ca} = \mathtt{f}$, which contradicts $S \models R_{02} = \mathtt{s}$. Proceeding to the second case:

$$S \models R_{02} = \mathtt{xs} \qquad \{\text{by hypothesis}\}$$

$$S_2 \models R_{02} = \mathtt{xs} \wedge T_1^{ca} = \mathtt{f} \wedge T_2^{ba} = \mathtt{f} \qquad \{\text{from (ii) and } S_2 = S[\mathtt{f}/T_2^{ba}]\}$$

but in that case $S_2 \models R_{02=\mathtt{s}} \Rightarrow T_1^{ca} = \mathtt{l} \wedge T_2^{ba} = \mathtt{l}$ now follows immediately since $S_2 \models R_{02} = \mathtt{xs}$ when $S \models R_{02} = \mathtt{xs}$. Hence, $S_2 \models \mathbf{RT}$ if $S \models \mathbf{RT}$. ■

To see how errors might be identified suppose that in (ii), for example, the variable T_1^{ac} was examined instead of T_1^{ca}. Case analysis would fail to find the contradiction when $R_{02} = \mathtt{s}$, so:

$$S \models R_{02} = \mathtt{s} \wedge T_1^{ca} = \mathtt{l} \wedge T_2^{ba} = \mathtt{l} \qquad \{\text{as before}\}$$

but clearly $S_2 \not\models R_{02=}\mathtt{s} \Rightarrow T_1^{ca} = \mathtt{l} \wedge T_2^{ba} = \mathtt{l}$, since $S_2 \models T_2^{ba} = \mathtt{f}$. In this case we therefore arrive at an assignment to the variables which, by (ii), leads to a successor state where the safety property does not hold. Errors in the conclusions of the rules may be similarly identified.

4.4.2 Route Request Data

It slightly more difficult to verify properties of the route request data since here several variables become instantiated by the assignments in the rules. The focus on R_{02} is maintained although this route does not bring to light all the issues (see Section 4.4.3, and Section 7.2 where overlaps are discussed).

Proposition 4.5 The route request rule for R_{02}

$$\begin{aligned}&\mathtt{*Q02\ if}\ P_1\,\mathtt{crf}, T_1^{ac}\,\mathtt{f}, T_2^{ab}\,\mathtt{f}\\ &\qquad\mathtt{then}\ R_{02}\,\mathtt{s}, P_1\,\mathtt{cr}, T_1^{ca}\,\mathtt{l}, T_2^{ba}\,\mathtt{l}\ \mathtt{\backslash\ .}\end{aligned}$$

preserves the invariance of $\mathbf{F} = \mathbf{MX} \wedge \mathbf{PT} \wedge \mathbf{RT}$. □

Proof The structure of the proof is much as that for Proposition 4.4. We show that if $S \models \mathbf{F}$ then $S' \models \mathbf{F}$. Since $\mathbf{F}$ is conjunctive, the proof breaks down into a number of components, $S \models \mathbf{F} \Rightarrow S' \models F_i$, each of which can be discharged independently.

From the definition of the Control (see Figure 3.4) it is clear that the following sequence of transitions leads from a state S to the successor state S':

$$\begin{aligned}
\mathsf{Control}(S) &= \mathsf{set}_{*\mathtt{Q02}}.(\mathsf{get}_{P_1}(\mathtt{cr}).\mathsf{Q02}_1(S) + \mathsf{get}_{P_1}(\mathtt{cn}).\mathsf{Q02}_2(S)) && \text{(iii)}\\
\mathsf{Q02}_2(S) &= \mathsf{get}_{T_1^{bc}}(\mathtt{f}).\mathsf{Q02}_3(S) + \mathsf{get}_{T_1^{bc}}(\mathtt{l}).\mathsf{Control}(S) && \text{(iv)}\\
\mathsf{Q02}_3(S) &= \mathsf{get}_{T_1^{cb}}(\mathtt{f}).\mathsf{Q02}_1(S) + \mathsf{get}_{T_1^{cb}}(\mathtt{l}).\mathsf{Control}(S) && \text{(v)}\\
\mathsf{Q02}_1(S) &= \mathsf{get}_{T_1^{ac}}(\mathtt{f}).\mathsf{Q02}_4(S) + \mathsf{get}_{T_1^{ac}}(\mathtt{l}).\mathsf{Control}(S) && \text{(vi)}\\
\mathsf{Q02}_4(S) &= \mathsf{get}_{T_2^{ab}}(\mathtt{f}).\mathsf{C02}(S) + \mathsf{get}_{T_2^{ab}}(\mathtt{l}).\mathsf{Control}(S) && \text{(vii)}\\
\mathsf{C02}(S) &= \overline{\mathsf{put}}_{R_{02}}(\mathtt{s}).\overline{\mathsf{put}}_{P_1}(\mathtt{cr}).\overline{\mathsf{put}}_{T_1^{ca}}(\mathtt{l}).\overline{\mathsf{put}}_{T_2^{ba}}(\mathtt{l}).\mathsf{Control}(S') && \text{(viii)}
\end{aligned}$$

where $S' = S[\mathtt{s}/R_{02}, \mathtt{cr}/P_1, \mathtt{l}/T_1^{ca}, \mathtt{l}/T_2^{ba}]$. The main difficulty here is in demonstrating that **MX** is invariant. Two terms in the conjunct require detailed assessment, corresponding to the mutual exclusion property for T_1 and T_2 respectively. Proceeding with the latter case:

$S \models \mathbf{MX}[T_2^{ab}, T_2^{ba}]$ {since $S \models \mathbf{MX}$}

$S \models \mathbf{MX}[T_2^{ab}, T_2^{ba}] \wedge T_2^{ab} = \mathtt{f}$ {from (vii)}

$S' \models T_2^{ba} = \mathtt{l} \wedge T_2^{ab} = \mathtt{f}$ {as $S' = S[\mathtt{s}/R_{02}, \mathtt{cr}/P_1, \mathtt{l}/T_1^{ca}, \mathtt{l}/T_2^{ba}]$}

The conclusion $S' \models \mathbf{MX}[T_2^{ab}, T_2^{ba}]$ follows immediately from the definition of **MX**. More difficult is the case where the sub-route traverses a points track section:

$S \models \mathbf{MX}[T_1^{ac}, T_1^{bc}, T_1^{ca}, T_1^{cb}]$ {since $S \models \mathbf{MX}$}

but now there are two cases to consider for from (iii) either $S \models P_1 = \mathtt{cr}$ or $S \models P_1 = \mathtt{cn}$. Proceeding with the latter:

$S \models P_1 = \mathtt{cn}$ {by hypothesis}

$S \models T_1^{cb} = \mathtt{f} \wedge T_1^{bc} = \mathtt{f}$ {from (v) and (iv)}

while on the other hand:

$S \models P_1 = \mathtt{cr}$ {by hypothesis}

$S \models T_1^{bc} = \mathtt{l} \vee T_1^{cb} = \mathtt{l} \Rightarrow P_1 = \mathtt{cn}$ {since $S \models \mathbf{PT}$}

$S \models T_1^{cb} = \mathtt{f} \wedge T_1^{bc} = \mathtt{f}$ {*modus tollens*}

Thus, in either case:

$S \models T_1^{ac} = \mathtt{f} \wedge T_1^{cb} = \mathtt{f} \wedge T_1^{bc} = \mathtt{f}$ {from (vi)}

$S' \models \mathbf{MX}[T_1^{ac}, T_1^{bc}, T_1^{ca}, T_1^{cb}]$ {as $S' = S[\mathtt{s}/R_{02}, \mathtt{cr}/P_1, \mathtt{l}/T_1^{ca}, \mathtt{l}/T_2^{ba}]$}

It follows therefore that $S' \models \mathbf{MX}$ if $S \models \mathbf{MX}$. It also follows, independently of $S \models \mathbf{RT}$, that the route property for R_{02} is invariant since plainly:

$S[\mathtt{s}/R_{02}, \mathtt{cr}/P_1, \mathtt{l}/T_1^{ca}, \mathtt{l}/T_2^{ba}] \models R_{02=}\mathtt{s} \Rightarrow T_1^{ca} = \mathtt{l} \wedge T_2^{ba} = \mathtt{l}$.

Finally the two safety properties for the points P_1 have to be checked since they are commanded to the reverse position by the rule. For the reverse direction:

$S' \models P_1 = \mathtt{cr} \wedge T_1^{ca} = \mathtt{l}$ {by definition of S'}

$S' \models T_1^{ac} = \mathtt{l} \vee T_1^{ca} = \mathtt{l} \Rightarrow P_1 = \mathtt{cr}$ {for any assignment to T_1^{ac}}

and for the normal direction of these points:

$S \models T_1^{bc} = \mathtt{f} \wedge T_1^{cb} = \mathtt{f}$ {case analysis as above}

$S \models T_1^{bc} = \mathtt{l} \vee T_1^{cb} = \mathtt{l} \Rightarrow P_1 = \mathtt{cn}$ {since $S \models \mathbf{PT}$}

The antecedent remains false in S' so the conclusion $S' \models \mathbf{PT}$ follows. Bringing the conjunctive parts together we can conclude therefore that $S \models \mathbf{F} \Rightarrow S' \models \mathbf{F}$. ■

In the above we used *modus tollens* to deduce $S \models T_1^{cb} = \mathtt{f} \wedge T_1^{bc} = \mathtt{f}$ assuming, in effect, that $P_1 = \mathtt{cn} \Rightarrow P_1 \neq \mathtt{cr}$. This, in turn, assumes that these SSI control variables are binary (which they are not). However, nothing in the model depends upon this assumption. Using the alternative characterisation of **PT** mentioned in Section 3.3.1 obviates the need to assume in the proof that $P_1 = \mathtt{cn} \Leftrightarrow P_1 \neq \mathtt{cr}$. Reformulating

the safety property appropriately we can instead use the hypothesis $S \models P_1 = \texttt{cr} \Rightarrow T_1^{cb} = \texttt{f} \wedge T_1^{bc} = \texttt{f}$ to arrive at the same conclusion. With the change to **PT** a slightly longer sequence of deductions is needed to show $S' \models$ **PT**. In detail:

$S \models P_1 = \texttt{cr} \Rightarrow T_1^{cb} = \texttt{f} \wedge T_1^{bc} = \texttt{f}$ {since $S \models$ **PT**}

$S \models P_1 = \texttt{cr}$ {by hypothesis}

$S \models T_1^{cb} = \texttt{f} \wedge T_1^{bc} = \texttt{f}$ {*modus ponens*}

$S' \models P_1 = \texttt{cr} \wedge T_1^{cb} = \texttt{f} \wedge T_1^{bc} = \texttt{f}$ {by definition of S'}

$S' \models P_1 = \texttt{cr} \Rightarrow T_1^{cb} = \texttt{f} \wedge T_1^{bc} = \texttt{f}$ {algebra}

On the other hand, the points may initially have been normal:

$S \models P_1 = \texttt{cn}$ {by hypothesis}

$S \models T_1^{cb} = \texttt{f} \wedge T_1^{bc} = \texttt{f}$ {by (iv) and (v)}

$S' \models P_1 = \texttt{cr} \wedge T_1^{cb} = \texttt{f} \wedge T_1^{bc} = \texttt{f}$ {by definition of S'}

$S' \models P_1 = \texttt{cr} \Rightarrow T_1^{cb} = \texttt{f} \wedge T_1^{bc} = \texttt{f}$ {algebra}

Finally for the points normal property:

$S \models P_1 = \texttt{cn} \Rightarrow T_1^{ac} = \texttt{f} \wedge T_1^{ca} = \texttt{f}$ {since $S \models$ **PT**}

$S' \models P_1 = \texttt{cn} \Rightarrow T_1^{ac} = \texttt{f} \wedge T_1^{ca} = \texttt{f}$ {since $S' \models T_1^{ca} = \texttt{l} \wedge P_1 = \texttt{cr}$}

This illustrates that the alternative characterisation of the safety property introduces no additional difficulty to the proof. Indeed, the proof simplified by the assumption $S \models P_1 = \texttt{cr} \Rightarrow T_1^{cb} = \texttt{f} \wedge T_1^{bc} = \texttt{f}$, so it is useful to prove *both* characterisations of **PT**. However, some properties are unprovable with the current infrastructure.

4.4.3 Unprovable Assertions

The uniformity exhibited by the sub-route release rules means that essentially the same proof method can be employed to verify safety properties for all sub-route release data. This degree of uniformity is not exhibited by the route request rules. For example, the rule for R_{02} is in some sense fully specified since the opposing sub-routes to all those along its length are tested in the precondition. This makes it easy to verify **MX** for each track section along the route. However, the rule for R_2 is not completely specified in this sense:

$$\texttt{*Q2 if } P_2\,\texttt{crf}, P_3\,\texttt{cnf}, T_4^{ac}\,\texttt{f}, T_7^{ab}\,\texttt{f}$$
$$\texttt{then } R_2\,\texttt{s}, P_2\,\texttt{cr}, P_3\,\texttt{cn}, T_4^{ca}\,\texttt{l}, T_6^{ca}\,\texttt{l}, T_7^{ba}\,\texttt{l} \texttt{ \textbackslash .}$$

The signalling principle here is that in checking for a route's availability it is only necessary to test the last conflicting sub-route on any opposing routes, together with the opposing sub-route over the berth track section of any *directly* opposing routes. To clarify, consider the routes depicted in Figure 4.2. For the routes from S_5 the last

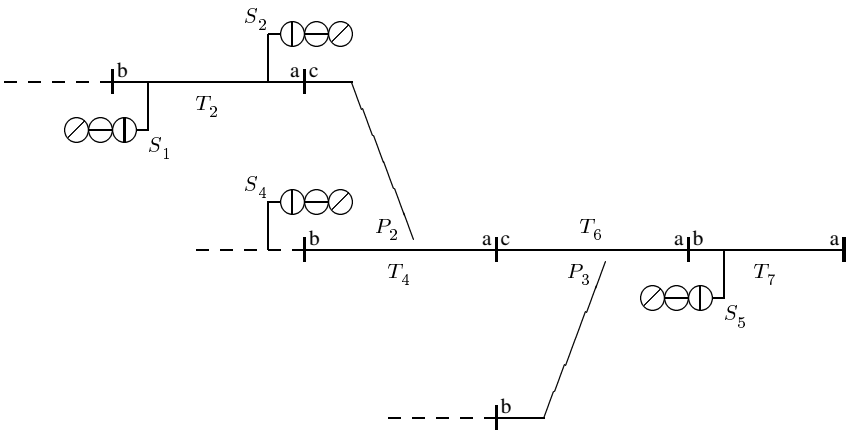


Figure 4.2: Routes R_2 and R_{51} from the scheme plan for WEST

conflicting sub-route on opposing routes R_2 and R_4 is T_6^{ca}; for R_{51}, the opposing sub-route in the berth track section of the directly opposing route is T_2^{ba}. Given that sub-routes are released sequentially along a route, the intermediate sub-routes do not need to be tested: T_6^{ac} should always be free when its successor (T_4^{ac}) is free.

To see where the paucity of information provided by the tests in `*Q2` leads to difficulty in the proof, consider the steps involved in showing that the **MX** property is preserved by this rule. There are three track sections to look at in detail, hence three terms in the invariant: $\mathbf{MX}[T_7^{ab}, T_7^{ba}]$, $\mathbf{MX}[T_4^{ab}, T_4^{ac}, T_4^{ba}, T_4^{ca}]$ and $\mathbf{MX}[T_6^{ab}, T_6^{ac}, T_6^{ba}, T_6^{ca}]$. The first two of these may be established just as in the proof sketched in the previous section; the latter is more problematic:

$$S \models \mathbf{MX}[T_6^{ab}, T_6^{ac}, T_6^{ba}, T_6^{ca}] \qquad \{\text{since } S \models \mathbf{MX}\}$$

By a case analysis as in the proof of Proposition 4.5, since $S \models P_3 = \texttt{cn}$ or $S \models P_3 = \texttt{cr}$, we find that in either case

$$S \models T_6^{ab} = \texttt{f} \wedge T_6^{ba} = \texttt{f}$$

$$S \models \mathbf{MX}[T_6^{ac}, T_6^{ca}] \qquad \{\text{simplifying}\}$$

Now the next state $S' \models T_6^{ca} = \texttt{l}$, so to establish $S' \models \mathbf{MX}$ we must first show $S \models T_6^{ac} = \texttt{f}$. But it is difficult to be unequivocal about this since possibly

$$S \models \mathbf{MX} \wedge T_6^{ca} = \texttt{f} \wedge T_6^{ac} = \texttt{l} \qquad (\dagger)$$

leading to an unsafe state in which both these sub-routes are locked. One cannot preclude this possibility from within the framework so far established. It is important to note that this is not simply an artefact of the formalism: the *real* SSI would enter the unsafe state if it were initially in this configuration. Seemingly, the system enters an unsafe state from a safe one. There is a clear objection, however, for while any state satisfying (†) may be "safe" with respect to the formula **F** it is not really safe in a wider sense. The sub-route T_6^{ac} should never be in the locked state when its successor T_4^{ac}

(or for that matter T_4^{ab}) is free: the free status of a sub-route depends on that of its predecessors. With regard to R_2 there are two options:

- If the sub-route release data are correct then by the rule for T_4^{ac} we can deduce $T_4^{ac} = \mathtt{f} \Rightarrow T_6^{ac} = \mathtt{f}$, which is enough to complete the arrested proof;
- Otherwise we can strengthen the invariant appropriately.

The former option was adopted when these ideas were first explored [74]. However, when we turn to formalising the mathematical arguments given in the preceding sections it is less problematic to adopt the second option. Discussion of precisely how we strengthen the invariant is deferred until Section 5.3.

4.5 From Rigorous to Formal Proofs

For the purposes of mechanically checking properties of the Geographic Data, the rigorous mathematical arguments of the kind presented above must be turned into fully formal proofs. Such proofs should never be performed by hand—rather, the objective here is to find an appropriate logic within which to formulate the proof steps. In this section we therefore survey some of the likely formalisms suggested by the literature on formal specification and verification. Specifically, we briefly reexamine the problem of proving safety properties of Geographic Data in the Temporal Logic of Actions, in the UNITY notation, and in the framework of Floyd-Hoare Logic. The invariance proof turns out to be very similar in each of these cases.

4.5.1 The Temporal Logic of Actions

Lamport's Temporal Logic of Actions [53] is described by its inventor simply as "mathematics plus box", the mathematics involved being the predicate calculus. The $\Box$ embellishment means *always*. TLA formulae are expressed in the syntax:

$$\Phi, \Psi ::= P \quad | \quad \neg\Psi \quad | \quad \Psi \wedge \Phi \quad | \quad \Box\mathcal{A} \quad | \quad \Box\Psi$$

Other common logical connectives are derived from these. Here P represents a *predicate* involving variables and numeric constants; $\mathcal{A}$ represents an *action*, these being formulae involving variables, primed variables and constants. The semantics of TLA are given in terms of infinite sequences of states (mappings from variables to values). A primed variable v' represents the value of the variable v in the 'next state'. The meaning of $\Box\Psi$ with respect to a sequence σ is that Ψ holds of all states in σ. In contrast, the meaning of $\Box\mathcal{A}$ is that $\mathcal{A}$ relates every pair of consecutive states in σ. Further constructs are introduced by Lamport to represent fair executions and stuttering states, but these need not concern us here.

A program is always represented in TLA by a formula of the form $I \wedge \Box\mathcal{A}$ where I is a predicate representing the initial state—*i.e.*, specifying the initial values of the program variables. $\mathcal{A}$ represents the possible steps that the program may take over time in modifying the initial state. To show that a program has property Φ one exhibits a proof of $I \wedge \Box\mathcal{A} \Rightarrow \Phi$. In particular, Φ may be a temporal formula: if $\Phi = \Box P$ and P is a predicate, then Φ is a safety (or invariance) property.

Geographic Data in TLA The *PRR* and *FOP* rules in the Geographic Data Language can be interpreted as actions in TLA. For example, the now familiar rules for R_{02} and its sub-routes may become:

$$(P_1 = \texttt{cr} \vee (T_1^{cb} = \texttt{f} \wedge T_1^{bc} = \texttt{f})) \wedge T_1^{ac} = \texttt{f} \wedge T_2^{ab} = \texttt{f} \Rightarrow$$
$$R_{02}{}' = \texttt{s} \wedge P_1{}' = \texttt{cr} \wedge T_1^{ca\prime} = \texttt{l} \wedge T_2^{ba\prime} = \texttt{l}$$
$$R_{02} = \texttt{xs} \wedge T_1 = \texttt{c} \Rightarrow T_1^{ca\prime} = \texttt{f}$$
$$T_1^{ca} = \texttt{f} \wedge T_2 = \texttt{c} \Rightarrow T_2^{ba\prime} = \texttt{f}$$

and so on. The formula $\mathcal{A}_{\text{ssi}}$, representing a single transition of the SSI, is expressed as the disjunction of terms such as these. This disjunction does not preclude the possibility that more than one of the rules may fire in a single step—in contrast to the CCS model which does. However, in accordance with the CCS model, $\mathcal{A}_{\text{ssi}}$ makes absolutely no commitment about the implementation of these rules.

Proving Safety The safety property is also expressed as a TLA formula—indeed, the predicate **F** in the previous section is already in the required form. To show that this is invariant, we must exhibit a proof that $I \wedge \Box\mathcal{A}_{\text{ssi}} \Rightarrow \Box\mathbf{F}$. Characterising the initial state of the system as 'any safe state' this obligation becomes: $\mathbf{F} \wedge \Box\mathcal{A}_{\text{ssi}} \Rightarrow \Box\mathbf{F}$. By the proof rules of TLA this is further transformed into the goal $\mathbf{F} \wedge \mathcal{A}_{\text{ssi}} \Rightarrow \mathbf{F}'$, where $\mathbf{F}'$ is the result of priming all the variables in **F**. Since $\mathcal{A}_{\text{ssi}}$ is a disjunctive formula, this proof naturally decomposes into a separate proof for each disjunct—*i.e.*, one proof for each rule in the *FOP* and *PRR* data:

$$\mathbf{F} \wedge (R_{02} = \texttt{xs} \wedge T_1 = \texttt{c} \Rightarrow T_1^{ca\prime} = \texttt{f}) \Rightarrow \mathbf{F}'$$

However, a difficulty emerges with this formulation since this formula is falsifiable. This is because the *action* only specifies that T_1^{ca} changes between states: we forgot to specify that all other control variables remain unchanged when this rule is executed. It is therefore necessary to strengthen each disjunct of $\mathcal{A}_{\text{ssi}}$. For example:

$$\mathbf{F} \wedge (R_{02} = \texttt{xs} \wedge T_1 = \texttt{c} \Rightarrow T_1^{ca\prime} = \texttt{f} \wedge \mathit{Unchanged}(\mathcal{D} - \{T_1^{ca}\})) \Rightarrow \mathbf{F}'$$

$\mathit{Unchanged}(\mathcal{D} - V)$ is a formula specifying that all variables in $\mathcal{D}$ other than those in V remain unchanged (as a result of the action). A similar term is needed in each disjunct, and with these additions the proof of the invariance of **F** can proceed.

Discussion The need to introduce the *unchanged* clauses is regrettable since these may add to the computational complexity of the formal (mechanical) proof. Additional complexity will be introduced when one considers modelling control flow. Lamport [53] gives examples where TLA formulae model the sequencing of actions in a program by introducing special 'control' variables (program counters). These model the control points in a (possibly parallel) program. The upshot of this is that for each action in the program one must additionally test for the control point. Although only one control variable will be introduced, this nevertheless introduces an artificial test to each rule in the data—the TLA representation of a program is therefore surprisingly concrete when one is obliged to specify which variables do and do not change in the execution of an action, and to explicitly keep track of the program counter. Lamport argues that TLA programs are in this sense completely specified, and sees this as a virtue paid for at the small price of added complexity in the specification. Unfortunately, the formulae arising from encoding Geographic Data in TLA are already very large, so any extraneous complexity introduced is certainly unwelcome.

4.5.2 Unity

Chandy and Misra's UNITY notation (this being an acronym for Unbounded Nondeterministic Iterative Transformation) has much in common with both TLA and Dijkstra's guarded command language. A UNITY program consists of a declaration of variables, their initial values, and a collection of guarded multi-assignments. UNITY's program execution model is that of an infinite sequence of (*state,command*) pairs: the next command to operate is chosen randomly, subject to the fairness constraint that every command is selected infinitely often. This fairness constraint aside, the program execution model (or operational semantics) is really no different from that of the CCS model presented in Section 3.2: there, upon hiding the visible actions, the recursive Control implements the unbounded nondeterministic iterative transformation of the state encoded in Image.

In their book [16] Chandy and Misra discuss a great many case-studies to introduce the notation and explain the development methodology, but they present UNITY's semantics only informally. This is regrettable since the logic they develop for reasoning about safety and liveness properties of programs evidently needs a precise foundation. Sanders [84] has shown that inconsistent deductions arise due to ambiguities in the definition of substitution; she avoids the problem by eliminating the Substitution Axiom and (in so doing) reducing the logic to linear temporal logic. More practically, Andersen *et al.* [1] have demonstrated that it is possible to formalise UNITY by interpreting the notation in higher-order logic in a way that avoids making inconsistent deductions.

The UNITY logic is a species of temporal logic which has assertions of the form $\{P\}\, s\, \{Q\}$ as its basis. These are reminiscent of Hoare triples (*cf.* Sections 4.5.3 and 5.2) where P is a predicate characterising the state in which the command s is executed, resulting in a state satisfying the predicate Q. If the initial condition that a program satisfies is I, a property P is invariant iff:

$$I \Rightarrow P \wedge \forall s.\{P\}\, s\, \{P\}$$

where quantification is over all program statements. In UNITY parlance the second conjunct asserts that P is *stable*. As has been seen, our main concern with the stability properties of Geographic Data.

Geographic Data in Unity It is straightforward to represent the Interlocking's control (*i.e.*, the data) as described in Chapter 3 in the UNITY notation:

program SSI
 declare $\mathcal{P}, \mathcal{T}, \mathcal{R}, \ldots$
 initially $\Big\|_{U \in \mathcal{U}} (U = \mathtt{f})$ $\Big\|_{R \in \mathcal{R}} (R = \mathtt{xs})$ ...
 always **F**
 assign $R_{02}, P_1, T_1^{ca}, T_2^{ba} := \mathtt{s}, \mathtt{cr}, \mathtt{l}, \mathtt{l}$
 if $(P_1 = \mathtt{cr} \vee (T_1^{cb} = \mathtt{f} \wedge T_1^{bc} = \mathtt{f})) \wedge T_1^{ac} = \mathtt{f} \wedge T_2^{ab} = \mathtt{f}$ $[\!]$
 $T_1^{ca} := \mathtt{f}$ **if** $R_{02} = \mathtt{xs} \wedge T_1 = \mathtt{c}$ $[\!]$
 $T_2^{ba} := \mathtt{f}$ **if** $T_1^{ca} = \mathtt{f} \wedge T_2 = \mathtt{c}$ $[\!]$ $\cdots$
end

The initial condition I is extracted from the *initially* section by interpreting $\|$ as conjunction; the *always* section specifies the invariant the program should maintain over all execution sequences—in this case the safety property **F**.

Proving Safety Stability of **F** amounts to showing, for each guarded command of the form $\tilde{x} := \tilde{e}$ **if** b, that $\mathbf{F} \wedge b \Rightarrow \mathbf{F}[\tilde{e}/\tilde{x}]$. In particular, to show that the sub-route release rule for T_1^{ca} ensures the stability of **F** one would prove

$$\mathbf{F} \wedge R_{02} = \mathtt{xs} \wedge T_1 = \mathtt{c} \Rightarrow \mathbf{F}[\mathtt{f}/T_1^{ca}]$$

which should be compared with the corresponding goal in Section 4.4.1. In that proof we started out with the goal of showing $S \models \mathbf{F} \Rightarrow S[\tilde{e}/\tilde{x}] \models \mathbf{F}$ and deduced that $S \models b$ from the transitions that must have occurred to allow the transformation in the state S. However, we might just as well have started out with the goal $S \models \mathbf{F} \wedge b \Rightarrow S[\tilde{e}/\tilde{x}] \models \mathbf{F}$, as per the partial tableau of Section 4.2.2.

Discussion Seldom does one encounter a programming notation which has been designed with such a flagrant disregard for control flow! For UNITY this is both a strength, in its targeted area of application, that of parallel program development, and a weakness in a setting where the sequential flow of control is an occasional concern. Nonetheless, it is possible to simulate sequential composition, by the same device as suggested for TLA programs, and there is a convincing parity between the UNITY formulation of the data correctness problem, and our earlier co-induction formulation: indeed, it is this same principle that underwrites the UNITY invariance proof.

4.5.3 Floyd-Hoare Logic

In the late 1960's Floyd [33], and Hoare [44], developed a logical notation for reasoning about simple imperative programs. At the heart of Hoare's *axiomatic* theory of partial correctness are assertions about program fragments of the form $\{P\}\, c\, \{Q\}$. As before, P characterises the state in which the program c is executed, and Q characterises the state of the memory of the machine when, and if, c terminates. The basic premise of Floyd-Hoare logic is the assignment axiom:

$$\text{ASS}\ \frac{}{\{P[e/x]\}\ x\ \texttt{:=}\ e\ \{P\}}$$

This is motivated by observing that if P holds in a state modified only by the assignment of the value of (the function) e to the variable x, then the predicate obtained by substituting e for all free occurrences of x in P holds before the assignment is made. It is easy to generalise this idea to parallel multi-assignment, like that seen in UNITY. Multi-assignments provide a convenient abstraction when it does not matter in which order a sequence of assignments is made.

Geographic Data in Floyd-Hoare Logic The Geographic Data Language is an interpreted language—the SSI control program providing the interpretation. From this perspective we may naturally consider the rules in the data to be commands in a simple language involving sequence, assignment and conditional jumping. As such

$$\texttt{*Q02 if}\ P_1\,\texttt{crf},T_1^{ac}\,\texttt{f},T_2^{ab}\,\texttt{f then}\ R_{02}\,\texttt{s},P_1\,\texttt{cr},T_1^{ca}\,\texttt{l},T_2^{ba}\,\texttt{l \textbackslash .}$$

clearly needs no further interpretation. Expanding the subroutine code inline (*i.e.*, the *PFM* test here, and the `@` specials elsewhere) obviates the need to define a program logic with subroutine handling facilities:

$$\texttt{*Q02 if (}P_1\texttt{=cr or }T_1^{bc}\texttt{=f},T_1^{cb}\texttt{=f)},T_1^{ac}\texttt{=f},T_2^{ab}\texttt{=f then ... \textbackslash .}$$

Proving Safety The safety analysis of the Geographic Data can proceed by exhibiting a proof of a theorem such as

$$\{\mathbf{F}\}\ \texttt{if}\ b\ \texttt{then}\ \tilde{x}\ \texttt{:=}\ \tilde{e}\ \{\mathbf{F}\}$$

for each data fragment in the *PRR* and *FOP* files. The multi-assignment axiom and the rules for one-armed conditional and strengthening in the precondition are used to derive the appropriate verification condition(s) from this assertion:

$$\text{IF}\ \frac{\text{PRE}\ \dfrac{\mathbf{F} \wedge b \Rightarrow \mathbf{F}[\tilde{e}/\tilde{x}] \qquad \text{ASS}\ \dfrac{}{\{\mathbf{F}[\tilde{e}/\tilde{x}]\}\ \tilde{x}\ \texttt{:=}\ \tilde{e}\ \{\mathbf{F}\}}}{\{\mathbf{F} \wedge b\}\ \tilde{x}\ \texttt{:=}\ \tilde{e}\ \{\mathbf{F}\}} \qquad \mathbf{F} \wedge \neg b \Rightarrow \mathbf{F}}{\{\mathbf{F}\}\ \texttt{if}\ b\ \texttt{then}\ \tilde{x}\ \texttt{:=}\ \tilde{e}\ \{\mathbf{F}\}}$$

This gives rise to two verification conditions—one is a trivial theorem of first-order logic, while the other is already too familiar.

Discussion It is not difficult to define an appropriate program logic for the general purpose conditional statements of the Geographic Data Language, nor for the 'specials' if they can be interpreted in these terms as in Sections 2.4 and 2.5. Floyd-Hoare logic—that is, the logic of partial correctness—is thus adequate for analysing invariant properties of Geographic Data. However, the logic is not well suited to analysing liveness or progress properties such as: "after initialisation, eventually a safe state will be reached." Using the logic of *total* correctness one may make such termination arguments, but generally one needs a dynamic or temporal logic in which to concisely express properties that speak about eventualities, and not just invariants.

4.6 Summary

The primary concern of this chapter has been to explore the nature of the invariance proof with a view to finding a simple and efficient means of mechanising the necessary reasoning. We began by looking at (local) model checking since it offers full automation and guarantees success one way or the other because the method is complete. The property either holds invariantly, or it does not. But given the astronomical sizes of the state spaces involved, model checking on its own is simply too expensive. Nevertheless, in exploring the structure of the proof tree for the given satisfaction problem, we found that it was possible to *fold* the structure into a partial tableau and transfer the proof to a simple induction argument.

That argument turns out to be an instance of the method of co-induction, and the invariance proof therefore reduces to the problem of demonstrating a closure property of a certain monotonic operator—*i.e.*, the Control. This dramatically improves the

tractability of the proof because it replaces the problem of showing that every state of the model is safe, with that of showing every fragment of the Geographic Data that is executed as a unit *preserves safety*.

Although the semantics of TLA (also those of UNITY) are given in terms of infinite sequences of states, whereas the semantics of the CCS model are given as a (finite) transition system, it turns out that co-induction is the unifying principle in the safety analysis. Recall that in the TLA setting invariance amounts to showing that all (finite or infinite) extensions of a sequence of safe states are safe. The proof step demonstrates that from a safe state the model can only reach one of a set of safe successors. Hence, by co-induction, all sequences extending from a safe state are safe.

Section 4.4 described some of the details of how to establish the appropriate verification conditions for the *PRR* and *FOP* data. Such detail is necessary for several reasons, not the least of which is that before proving something with a theorem prover's support we must first *understand* the proof. Moreover, as demonstrated by the 'failed' proof in Section 4.4.3, it is important to know whether there is in fact enough information to complete the proof envisaged. Careful analysis indicated that this was not the case for the given formulation of the problem, and revealed that the safety property defined in Chapter 3 was too weak in general. This weakness will be repaired in the next chapter.

Mathematical sophistication is not required to prove safety in interlocking data. In a sense, this is entirely as it should be since the reason why a system is safe ought to be sufficiently simple to be convincing. However, mathematical insight and a detailed knowledge of the problem at hand are needed to select the right representation within which to conduct the formal analysis. The discussion in Section 4.5 reveals that there is no obviously best choice in this matter. TLA is an expressive logic within which one can capture not only safety properties of systems, but also properties such as liveness and fairness which express eventualities. But TLA imposes penalties too, since it becomes inconvenient to reason about simple sequencing of events. The CCS and μ-calculus framework with which the analysis began does not carry this penalty—in process algebra, we may model functional aspects of the SSI's control with arbitrary precision. However, model checking is enormously computationally expensive.

It is straightforward to define a program logic for the general purpose conditional statements of the Geographic Data Language, slightly less so for the 'specials'. The data preparation guide [9] observes (but without justification) that the specials never in fact need to be used because one can express all the required signalling functions in the language of sequential and guarded commands. The specials are only designed to speed the real-time functioning of the SSI. As this is irrelevant from the (functional) verification standpoint we may assume that they are always expanded into equivalent conditional code. However, it *is* of course relevant that the SSI control interpreter

and the translation mechanism agree on the semantics of the specials. Justification for this position was given in Chapter 2, but however we set about verifying properties of Geographic Data, a proper treatment of the specials has to be taken into consideration.

By insisting that the safety property holds both before the block of code is executed as well as after it has terminated, other data, including *OPT* data, may be treated in a manner similar to the guarded command illustrated earlier. However, where commands are in sequence it is necessary to specify or *assert* [35] the weakest condition that should hold at the intermediate states. Since the safety critical states in the evolution of the system are those at which the command interpreter is evaluating the guard of a command (to make the next signalling decision), it is appropriate that the weakest condition here be the safety property **F**. These were identified in Chapter 3 as the control points in the model. At other intermediate states (and these will only be between updates to the image of the railway) the weakest condition will be *true*. This justifies treating sequences of assignments as multi-assignments: it is reasonable since only constant expressions are allowed at the level of Geographic Data (the interpreter updates counters, *etc.*).

Not without some misgivings we settle, therefore, for conducting the invariance proof in Floyd-Hoare logic. Firstly, this notation has an intuitive appeal likely to be more attractive to non-specialists in formal methods, but who may nevertheless have to certify that the results of our analysis present a convincing argument that the data are safe. Secondly, we retain greater control over the structure of the proof because the model naturally permits sequencing of commands. Thirdly, the proof system of Floyd-Hoare logic is compositional. On the one hand, compositionality means that we do not have to treat every rule in the Geographic Data as a unit, but can rather examine its structure (intermediate states). On the other hand, compositionality can lead to shortcuts in proofs if the same formal argument is needed in several places—for instance, if some code is used in several contexts, or on the disjunctive branches of a complex proof.

Later, in Chapter 6 where issues raised by the interactions between Interlockings are considered, there will be some cause to review this decision to use a simple program logic since the properties of interest there are not readily expressible in the Floyd-Hoare logic of partial correctness. In that chapter we shall return to the richer modal μ-calculus. Meanwhile we have not only to formalise Floyd-Hoare logic in an appropriate environment, but also have to construct a program that will implement the proof strategies worked out in Section 4.4. That is the subject of the following chapter.

Chapter 5

A Formal Theory of the Geographic Data Language

In this chapter a fully automatic means of checking safety properties of Geographic Data is recovered. This is achieved by implementing the proof ideas discussed in the preceding chapter within the formal constraints of a deductive reasoning system. After a brief introduction, we begin in Section 5.2 by formalising the semantics of the Geographic Data Language in higher-order logic, and obtain a program logic as a collection of derived inference rules. In Section 5.3 a simple theory of Geographic Data invariants is described, which is used in Section 5.4 to formalise the invariance proofs sketched earlier for the *PRR* and *FOP* data. In Section 5.5 we assess the computational complexity of the approach, and describe some heuristics to decompose the (quadratic time) global proof into more readily tractable local constituents.

5.1 Introduction

The typical proofs sketched in Section 4.4 are too laborious to be done reliably by hand. Yet it is essential to conduct some proofs in this way if one is to devise special purpose proof schemas to handle various classes of Geographic Data. The challenge is then to devise a method of mechanically verifying safety properties of GDL programs. Ideally a fully automated tool should be provided since, while railway signalling is in some sense a logical discipline, we should not require railway signalling engineers to be also adept at formal proof.

Rigorous mathematical proofs are not necessarily easy to transcribe into the purely formal, symbolic manipulations mandated by automated proof checkers and the *formal systems* that underly them. Some experimentation is necessary, for which an *interactive* theorem prover is needed, equipped with a *metalanguage* in which to express the derived proof procedures for subsequent reuse. Milner called these *tactics* [60] in the context of automated formal proof.

The HOL system [34] meets this requirement, although not uniquely. HOL supports a variety of higher-order logic, a type theory derived from Church's typed lambda calculus [17], and the Logic of Computable Functions [36] from which it inherits a polymorphic type discipline. Some features of the HOL system which make it a suitable vehicle with which to implement a Geographic Data theorem prover include:

- Mechanisms to support both forward proof by means of primitive and derived inference rules of higher-order logic, and backward, goal orientated proof by means of tactics under user or program control.
- A ready collection of rules and tactics and a language of *tacticals* with which to combine them for specialised applications. Rules, tactics and tacticals are just programs written in ML, the theorem prover's metalanguage.
- ML's own type discipline means values representing theorems of higher-order logic can only be obtained by applying the primitive inference rules of the logic. One may write arbitrarily proof procedures as ML programs, and the system's type security ensures that only valid theorems result.
- HOL is an open programming system so one can provide parsers, unparsers, and GDL syntax checkers, entirely within the ML system. Otherwise one can run the theorem prover as a client in a larger environment and communicate with the command loop through a variety of mechanisms [94, 92].
- The HOL system and its logic are stable, and widely used in academic and industrial institutions. A strong user base can be taken to mean that the system has progressed from an experimental platform to a proven technology.

A growing number of freely and commercially available proof systems offer similar functionality to HOL. Of these, Isabelle (Paulson [79]) and PVS (Shankar *et al.* [77]) warrant mention—the former because it is a generic theorem prover in which one can readily encode a specialised theory of Geographic Data like that described in here; the latter because it offers very powerful decision procedures for first-order logic and arithmetic (which HOL currently lacks). With PVS however, adapting the tool to application specific tasks is difficult since it is not provided with an open programming environment. Commercial variants on the theme in which one could directly implement the method discussed below include ProofPower (International Computers Ltd.) and LAMBDA (Abstract Hardware Ltd.).

The approach taken to providing a Geographic Data theorem prover is based on Gordon's experimental embedding of program logics in HOL [35]. We begin in Section 5.2 below by formalising a denotational semantics for the Geographic Data Language, and exploit the expressive power of higher-order logic to represent the semantics

directly in the theorem prover. Such an approach would not be possible in a first-order logic, for example. The rules and axioms of Floyd-Hoare logic are then mechanically derived from the formal theory of the semantics of GDL. This guarantees the validity of the program logic.

The theory of the embedded programming language developed differs from Gordon's **while** language, notably because there is no need for the **while** construct in Geographic Data. However, Gordon defines a language having only the natural numbers as data, and this should be generalised in order to reason about the concrete datatypes for points, signals, and so on. The theory described in Section 5.2 is in fact *polymorphic*: states are modelled by functions from a concrete domain of program identifiers to an unspecified data domain represented by a *type variable*. The image of the SSI is represented by a collection of such functions.

The effort of constructing a polymorphic theory pays in a prototype verification tool since we may readily experiment with different representations of the data. The theory is instantiated in Section 5.3 when particular representations for the datatypes are chosen: routes, sub-routes and track circuits are modelled as Boolean variables, but points are treated more elaborately. In accordance with this representation we next develop a theory of the invariants discussed earlier, taking care to repair the weakness in **RT** identified in Section 4.4.3. The precise formulation of this property turns out to be delicate.

A tactic in the HOL system is a function that given a goal to prove will return a collection of (simpler) subgoals together with a *validation*. A validation is an ML program that given a proof for each subgoal will yield a proof of the original goal. The composition of these functions is a proof of the initial conjecture given to the system, the result of which is a theorem. Now given a partial correctness specification $\{\mathbf{F}\}c\{\mathbf{F}\}$, the program uses a combination of tactics to decompose c into a number of verification conditions according to the syntactic structure. The validations of these tactics are the derived rules of Floyd-Hoare logic. Thus the problem addressed in Section 5.4 is how to prove the verification conditions which arise from the sub-route release and panel route request rules. Two tactics are offered that correspond to the demonstrations in Sections 4.4.1 and 4.4.2.

In Section 5.5 the computational complexity of the proof method is examined. It turns out that the size of the invariant $\mathbf{F}$ (*i.e.*, the number of conjuncts) is proportional to the size—in terms of the number of rules—of the verification task. This gives rise to quadratic time complexity in proving $\{\mathbf{F}\}c\{\mathbf{F}\}$ for all c. Space requirements are linear. The quadratic time complexity arises because we insist on establishing that *local* safety properties of the Geographic Data hold *globally*. In Section 5.5 this issue is addressed by considering heuristics to decompose the verification task. In fact we decompose

the proofs. These heuristics are intended to be implemented in ML programs which, together with cosmetic but desirable utilities such as parsers and pretty-printers to hide the HOL syntax, would provide the foundation of a mature tool for analysing safety properties of Geographic Data.

5.2 Geographic Data in Higher-order Logic

The Geographic Data Language is an interpreted programming language. It is also a weak language whose syntactic constructs include only simple assignment (of constants to variables), sequence, one- and two-armed conditionals and a switch construct. The datatypes over which the control structures operate are of a fixed format as described in Section 2.2. There is, however, a primitive subroutine mechanism which is mainly used in the *PRR* and *PFM* data. The `@` directive diverts the interpreter to a block of code that may be common to several routes: in the context of a test this will be an *evaluation set*, having no side-effect; in the context of a command this will be an *execution set*.

In checking properties of Geographic Data one must always expand evaluation sets inline since their adequacy is context dependent. For execution sets one may optionally seek to verify that the common blocks of data are independently safe. This offers a potential shortcut in proofs where a common block is referenced—because one can appeal to a pre-proved theorem, like $\{\mathbf{F}\}\, c\, \{\mathbf{F}\}$, instead of reproving the same theorem several times. But this will be left for future optimisation: instead of formalising the semantics of jumps explicitly the subroutine code will be assumed to have been expanded inline before the formal verification proceeds. For GDL this never alters the meaning of the program.

The absence of a looping construct or any form of nondeterminism means that the semantics of the Geographic Data Language are easy to define formally—although the SSI's designers have only provided a formal statement of the language's syntax. In putting forward a formal semantics below it must be remembered therefore that it is the control interpreter itself which *defines* the language. Consequently, the formal semantics can only be faithful to the informal description given in [9].

5.2.1 A Simple Imperative Language

The simple imperative language of **while** programs forms the introductory basis of a number of text books on the subject of programming language semantics, such as Tennent's [93], but the language formalised here is simpler even than this since it omits the looping constructs. For the moment suppose that values (**Val**) in the language will be of several primitive types, including at least truth values and natural numbers. The

appropriate domains of interpretation are

$$\begin{aligned} \mathcal{B} &= \{\textbf{true}, \textbf{false}\} \quad \text{and} \\ \mathcal{N} &= \{0, 1, 2, \ldots\} \end{aligned}$$

as expected. Expressions will be of these types, but no particular expression language here is assumed here. Let e, b, c and g, be metavariables ranging over the syntactic categories of expressions, Boolean expressions and commands respectively, with $x, y, \ldots$ representing variable identifiers (**Var**). The abstract syntax of commands is (see Section 2.4.1) can be summarised thus:

$$c \ ::= \ x \texttt{ := } e \mid c_1 \texttt{ ; } c_2 \mid \texttt{if } b \texttt{ then } c \mid \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \mid \texttt{skip}$$

Formally we need `skip`, the command that does nothing, because the concrete syntax admits an empty command list. The case construction of Section 2.3.2 can also be introduced (indeed, it is necessary to do so if the verification tool is to produce meaningful error messages) with the clauses:

$$\begin{aligned} c &::= \cdots \mid \texttt{(if } b \texttt{ then } c \texttt{ or } g \texttt{)} \\ g &::= c \mid \texttt{if } b \texttt{ then } c \texttt{ or } g \end{aligned}$$

but its semantics is just that of the conditional. Only the two-armed conditional is needed (given `skip`) for the theory development, but the one-armed conditional is more prevalent in Geographic Data, so it is retained as a primitive.

The semantics described here are functional (and presented in slightly different style to Section 2.4): the *meaning* of a command in this language will be a function between program states, a state being a mapping from variables to an appropriate domain of values. The meaning of an expression will be a function from states to values (of appropriate type):

$$\begin{aligned} \textbf{State} &= \textbf{Var} \longrightarrow \textbf{Val} \\ \mathcal{E}[\![\cdot]\!]_\theta &: \textbf{State} \longrightarrow \textbf{Val}_\theta \\ \mathcal{C}[\![\cdot]\!] &: \textbf{State} \longrightarrow \textbf{State} \end{aligned}$$

The semantics of commands is summarised in Figure 5.1. Note in particular that the assignment x `:=` e, when evaluated in a state s, yields a new state that differs from the old only in that x now maps to the value of e in s. In fact there will be several versions of assignment, corresponding to the differing expression types in the language. In the displayed semantics a type subscript is used to distinguish phrases (expressions) of differing types: $\mathcal{E}[\![e]\!]_\theta$ evaluates e to yield a value of type θ, which is then bound in s to an identifier of the same type.

$$\mathcal{C}[\![\texttt{skip}]\!]\, s = s$$
$$\mathcal{C}[\![x \texttt{ := } e]\!]\, s = s[\mathcal{E}[\![e]\!]_\theta\, s \,/\, x_\theta]$$
$$\mathcal{C}[\![c_1 \texttt{ ; } c_2]\!]\, s = \mathcal{C}[\![c_2]\!](\mathcal{C}[\![c_1]\!]\, s)$$
$$\mathcal{C}[\![\texttt{if } b \texttt{ then } c]\!]\, s = \text{if } \mathcal{E}[\![b]\!]_{\mathcal{B}}\, s = \textbf{true} \text{ then } \mathcal{C}[\![c]\!]\, s \text{ else } s$$
$$\mathcal{C}[\![\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2]\!]\, s = \text{if } \mathcal{E}[\![b]\!]_{\mathcal{B}}\, s = \textbf{true} \text{ then } \mathcal{C}[\![c_1]\!]\, s \text{ else } \mathcal{C}[\![c_2]\!]\, s$$
$$\mathcal{C}[\![\texttt{(if } b \texttt{ then } c \texttt{ or } g \texttt{)}]\!]\, s = \mathcal{G}[\![\texttt{if } b \texttt{ then } c \texttt{ or } g]\!]\, s$$
$$\mathcal{G}[\![c]\!]\, s = \mathcal{C}[\![c]\!]\, s$$
$$\mathcal{G}[\![\texttt{if } b \texttt{ then } c \texttt{ or } g]\!]\, s = \text{if } \mathcal{E}[\![b]\!]_{\mathcal{B}}\, s = \textbf{true} \text{ then } \mathcal{C}[\![c]\!]\, s \text{ else } \mathcal{G}[\![g]\!]\, s$$

Figure 5.1: Denotational Semantics of Geographic Data Language Commands

In implementing this language in HOL it will be more appropriate to adopt a relational style of presentation. Generally, semantic equations such as those displayed above give rise to *partial* functions, which are difficult to represent or reason about in the logic of the HOL system. It turns out, however, that the semantics of GDL define a total function: every well formed phrase of the language denotes a value—essentially because all commands are finite, and the expression evaluation function is total (expressions are constant functions, in fact). But it is still less problematic in HOL to adopt the relational style of presentation. Note that if $[\![c]\!]$ is the denotation of c in the relational presentation, $(s, s') \in [\![c]\!]$ just in case $\mathcal{C}[\![c]\!]\, s = s'$.

5.2.2 Semantics in Higher-order Logic

We follow Gordon [35] closely in formalising the semantics of the Geographic Data Language in HOL. However, the language Gordon described had only a single datatype, the natural numbers, and while this is adequate the restriction is inconvenient since it will (ultimately) be desirable to represent the concrete datatypes arising in Geographic Data. This immediately raises the problem of modelling a state in the domain

$$\textbf{State} : \textbf{Var} \longrightarrow \mathcal{B} + \mathcal{N} + \cdots$$

since it is difficult, in a straightforward implementation, to represent a function defined over disjoint domain or codomain. The logic of the HOL system is a modest variant of Church's simple type theory and does not provide *sum types*. One way forward is to use the clever embedding described by Melham [58] to define variant records in the logic, and thereby simulate the above function domain. The '$+$' type operator is part of the HOL basis, but the disadvantage of using Melham's sum types is that they would require *program* variables to have types $\mathcal{B} + \mathcal{N} + \cdots$, *etc.*, which is somewhat unnatural.

The simpler approach is taken here of splitting the state into the product of several functions. To fix ideas we introduce the type abbreviation

$$\mathit{state} = (\mathit{string} \to \alpha) \times (\mathit{string} \to \beta)$$

where *string* is the predefined type of (ASCII) strings and α and β are type variables in higher-order logic. In this theory, program variables (identifiers) will be represented simply by strings. Clearly one could add further state components in a similar way, *e.g.*

$$\mathit{state} = (\mathit{string} \to \alpha) \times (\mathit{string} \to \beta) \times (\mathit{string} \to \gamma) \cdots$$

but just the two will suffice for our current purposes. The theory of the semantics of the Geographic Data Language will be instantiated later, in Section 5.3, by supplying concrete types—such as *num*, *bool*, or *four-bit-word*—in place of the type variables appearing here. The advantage gained by constructing a polymorphic theory in the prototype verification tool is that one may easily experiment with different representations of the data.

Now the semantic evaluation functions (for expressions) will be represented in the HOL theory by functions of type $\mathit{state} \to \alpha$ and $\mathit{state} \to \beta$ respectively. Predicates are functions $\mathit{state} \to \mathit{bool}$. Then, in order to bind a sequence of values into the appropriate state component we define constants Bind and BindSeq:

$$\vdash \forall\, x\, v\, (g{:}\,\mathit{state} \to \gamma).\, \mathsf{Bind}\, x\, v\, g = \lambda z.\, (z = x \to v \mid g\, x)$$

$$\vdash (\forall\, (s{:}\,\mathit{state})\, X\, (g{:}\,\mathit{state} \to \gamma).\, \mathsf{BindSeq}\, s\, X\, \mathsf{nil}\, g = g) \wedge$$

$$(\forall\, s\, X\, e\, E\, g.$$

$$\mathsf{BindSeq}\, s\, X\, (\mathsf{cons}\, e\, E)\, g = \mathsf{Bind}\, (\mathsf{hd}\, X)\, (e\, s)\, (\mathsf{BindSeq}\, s\, (\mathsf{tl}\, X)\, E\, g))$$

The first of these defines the Bind operator which binds a value v of type γ to identifier x in state (component) g. The result is a new state, a function in higher-order logic from identifiers to values. The type variables α and β (and γ, which must match one of these) appearing in the definitions are universally quantified—so the operators are well defined when any concrete types are instantiated in their place. Note that $(a \to b \mid c)$ is HOL syntax for 'if a then b else c' which is well typed only if a is a proposition and b and c are of the same logical type. Also note that function application associates to the left.

The first argument to BindSeq is the state (s) in which the expressions E are to be evaluated: the last argument (g) is the state component into which the variables are to be bound. The constants nil and cons are constructors of the type of polymorphic lists; hd and tl being the obvious list operations. BindSeq is defined by unwinding the recursion along the second list, of expressions, but it would seem to be more natural instead to write:

$$\mathsf{BindSeq}\, s\, \mathsf{nil}\, \mathsf{nil}\, g = g \wedge$$

$$\mathsf{BindSeq}\, s\, (\mathsf{cons}\, x\, X)\, (\mathsf{cons}\, e\, E)\, g = \mathsf{Bind}\, x\, (e\ s)\, (\mathsf{BindSeq}\, s\, X\, E\, g)$$

However, this formulation introduces domain equations that the HOL system cannot solve when it attempts to prove that the definition is logically sound.

The definitions above illustrate that the theory of the Geographic Data Language has parent theories of lists, strings, *etc.*. In the sequel the syntax of lists will be simplified in order to make the formal definitions more readable, using $[\,]$ for the empty list, and $[h \mid t]$ for $\mathsf{cons}\, h\, t$. The syntax $[a, b]$ will occasionally be used in lieu of $\mathsf{cons}\, a\, (\mathsf{cons}\, b\, \mathsf{nil})$, and when they appear, strings will be delimited by quotes (`'·'`). We may then prove some simple theorems about binding:

$$\vdash \mathsf{BindSeq}\, s\, [\,]\, [\,]\, g\, x = g\, x$$

$$\vdash \mathsf{BindSeq}\, s\, [x \mid X]\, [e \mid E]\, g\, x = e\ s$$

$$\vdash \neg(x = y) \Rightarrow (\mathsf{BindSeq}\, s\, [x \mid X]\, [e \mid E]\, g\, y = \mathsf{BindSeq}\, s\, X\, E\, g\, y)$$

The free variables in these theorems are implicitly universally quantified. In particular, the free type variables are universally quantified—so the theorems remain true under any type instantiation. In the sequel, universal quantification and explicit type annotation will often be elided.

The constant BindSeq simultaneously binds a sequence of values—in order to define parallel multi-assignment which generalises the single assignment seen earlier:

$$\vdash \forall\, X\, E\, Y\, F\, s\, s'.\mathsf{Ass}(X, E, Y, F)\, (s, s') =$$

$$(s' = (\mathsf{BindSeq}\, s\, X\, E\, (\mathsf{fst}\, s)\,,\ \mathsf{BindSeq}\, s\, Y\, F\, (\mathsf{snd}\, s)))$$

Operators fst and snd project the components of the pair s. Fortunately the user of the system never need be aware of the HOL presentation of the object language syntax. Although the aim is for full automation, where command loop interaction with the tool is necessary one can modify the parser to admit GDL syntax directly (as in [35]).

In these semantics a command has logical type *state* $\times$ *state* $\rightarrow$ *bool*. This means that the semantics of the embedded language is represented by a relation—as promised at the end of the preceding section. The definition above asserts that s' is the result of binding the values of the expressions E to identifiers X in the first component, and F to Y in the second. Where assignments are all of the form x `:=` k, for *constant* k, x_1, x_2 `:=` k_1, k_2 is equivalent to x_1 `:=` k_1 `;` x_2 `:=` k_2 as long as x_1 and x_2 are distinct. This treatment means that only one version of assignment needs to be defined, rather than one for each datatype in the language. Consequently only one version of the assignment axiom needs to be derived. (However, this is only a matter of style: the advantage gained from the multi-assignment primitive is only tenuous, and none

of the subsequent developments depend critically on this particular formulation of assignment.)

The other command forms of the Geographic Data Language are represented more naturally in higher-order logic:

$$\vdash \forall\, s\, s'.\, \mathsf{Skip}\,(s, s') = (s = s')$$

$$\vdash \forall\, c\, c' s\, s'.\, \mathsf{Seq}\,(c, c')\,(s, s') = \exists s''.c(s, s'') \wedge c'(s'', s')$$

$$\vdash \forall\, b\, c\, s\, s'.\, \mathsf{If}\,(b, c)\,(s, s') = (b\, s \rightarrow c(s, s')\ \mid\ s = s')$$

$$\vdash \forall\, b\, c\, c'\, s\, s'.\, \mathsf{Ite}\,(b, c, c')\,(s, s') = (b\, s \rightarrow c(s, s')\ \mid\ c'(s, s'))$$

This conceals the fact that s, s' are pairs of functions. These definitions, and the theorems concerning binding, complete the formal (HOL) theory of the semantics.

5.2.3 Hoare Logic: Rules and Tactics

Again, the details here follow Gordon [35] so only the delicate parts of the embedding are indicated, and where the more general theory introduces subtleties. The approach is to derive the rules and axioms of partial correctness specifications from the theory of the denotational semantics. The program logic, or axiomatic semantics of the programming language [93], was described in Section 4.5.3. The assertion $\{P\}\, c\, \{Q\}$ may be represented in higher-order logic by introducing a new constant Spec

$$\vdash \forall\, (c{:}\, \mathit{state} \times \mathit{state} \rightarrow \mathit{bool})\, p\, q.\, \mathsf{Spec}(p, c, q) = \forall\, s\, s'.p\, s \wedge c(s, s') \Rightarrow q\, s'$$

where the pre- and post-conditions here are state predicates. Under this scheme an assertion about an SSI control variable, say $\{T_2 = v\}$, is denoted by a HOL term like $\lambda s{:}\, \mathit{state}.\mathsf{snd}\, s\texttt{'T2'} = v$. In the following let $[\![P]\!]$ be the denotation in HOL of the predicate P (so that $[\![T_2 = v]\!]$ abbreviates $\lambda s.\mathsf{snd}\, s\texttt{'T2'} = v$, say).

5.2.3.1 Derived Rules of Floyd-Hoare Logic

From the definitions outlined above one can routinely prove some theorems about Spec:

$$\vdash (\forall\, s.\, p' s \Rightarrow p\, s) \wedge \mathsf{Spec}(p, c, q) \Rightarrow \mathsf{Spec}(p', c, q)$$

$$\vdash (\forall\, s.\, q\, s \Rightarrow q'\, s) \wedge \mathsf{Spec}(p, c, q) \Rightarrow \mathsf{Spec}(p, c, q')$$

$$\vdash \mathsf{Spec}((\lambda s.\, p\,(\mathsf{BindSeq}\, s\, x\, e\,(\mathsf{fst}\, s), \mathsf{BindSeq}\, s\, y\, f\,(\mathsf{snd}\, s))), \mathsf{Ass}(x, e, y, f), p)$$

$$\vdash \mathsf{Spec}(p, c, r) \wedge \mathsf{Spec}(r, c', q) \Rightarrow \mathsf{Spec}(p, \mathsf{Seq}(c, c'), q)$$

$$\vdash \mathsf{Spec}((\lambda s.\, p\, s \wedge b\, s), c, q) \wedge (\forall\, s.\, p\, s\ \wedge \neg(b\, s) \Rightarrow q\, s) \Rightarrow \mathsf{Spec}(p, \mathsf{If}(b, c), q)$$

In this way the rules of Floyd-Hoare logic become derived rules of higher-order logic.

The other constructs of the embedded language are treated similarly. The last of the above theorems is the foundation from which the `IF` rule is derived:

$$\texttt{IF}\ \frac{\vdash \{P \wedge b\}\, c\, \{Q\} \qquad \vdash P \wedge \neg b \Rightarrow Q}{\vdash \{P\}\ \texttt{if}\ b\ \texttt{then}\ c\, \{Q\}}$$

Note, however, that the second antecedent is a theorem of higher-order logic, while the former is a theorem of Hoare logic. The distinction is delicate: the program variables appearing in $\{P \wedge b\}$ are quoted, while they appear as (unquoted) logic variables in $P \wedge \neg b \Rightarrow Q$. As the free variables in the theorems above are universally quantified we can specialise

$$\vdash \mathsf{Spec}((\lambda s.\, p\, s \wedge b\, s), c, q) \wedge (\forall\, s.\, p\, s\ \wedge \neg(b\, s) \Rightarrow q\, s) \Rightarrow \mathsf{Spec}(p, \mathsf{If}(b, c), q)$$

to:

$$\vdash \mathsf{Spec}((\lambda s.\, [\![P]\!] s \wedge [\![b]\!] s), [\![c]\!], [\![Q]\!]) \wedge (\forall\, s.\, [\![P]\!] s\ \wedge \neg([\![b]\!] s) \Rightarrow [\![Q]\!] s)$$
$$\Rightarrow \mathsf{Spec}([\![P]\!], \mathsf{If}([\![b]\!], [\![c]\!]), [\![Q]\!])$$

Since $(\lambda s.\, [\![P]\!] s \wedge [\![b]\!] s) \equiv_\beta [\![P \wedge b]\!]$, this is beta-convertible to

$$\vdash \mathsf{Spec}([\![P \wedge b]\!], [\![c]\!], [\![Q]\!]) \wedge (\forall\, s.\, [\![P]\!] s\ \wedge \neg([\![b]\!] s) \Rightarrow [\![Q]\!] s)$$
$$\Rightarrow \mathsf{Spec}([\![P]\!], \mathsf{If}([\![b]\!], [\![c]\!]), [\![Q]\!])$$

and the `IF` rule follows from this as long as $\vdash \forall\, s.\, [\![P]\!] s\ \wedge \neg([\![b]\!] s) \Rightarrow [\![Q]\!] s$ can be derived from (the verification condition) $\vdash P \wedge \neg b \Rightarrow Q$. To see that this is the case suppose, without loss of generality, that the free (program) variables in P, b and Q of type α are contained in $A_1, \ldots A_m$, and those of type β are contained in $B_1, \ldots B_n$. Then denote these explicitly in P by writing $P[A_1, \ldots A_m, B_1, \ldots B_n]$, *etc.*. The variables $A_1, \ldots B_n$ are free in

$$\vdash P[A_1, \ldots B_n] \wedge \neg b[A_1, \ldots B_n] \Rightarrow Q[A_1, \ldots B_n]$$

and so the A_i and B_j can be instantiated with $\mathsf{fst}\, s\texttt{'Ai'}$ and $\mathsf{snd}\, s\texttt{'Bj'}$ respectively, and the free variable s: *state* generalised to obtain

$$\vdash \forall\, s. P[\mathsf{fst}\, s\texttt{'A1'}, \ldots \mathsf{snd}\, s\texttt{'Bn'}] \wedge \neg b[\mathsf{fst}\, s\texttt{'A1'}, \ldots \mathsf{snd}\, s\texttt{'Bn'}]$$
$$\Rightarrow Q[\mathsf{fst}\, s\texttt{'A1'}, \ldots \mathsf{snd}\, s\texttt{'Bn'}]$$

which is beta-convertible to:

$$\vdash \forall\, s. (\lambda s. P[\mathsf{fst}\, s\texttt{'A1'}, \ldots \mathsf{snd}\, s\texttt{'Bn'}])s \wedge \neg((\lambda s. b[\mathsf{fst}\, s\texttt{'A1'}, \ldots \mathsf{snd}\, s\texttt{'Bn'}])s)$$
$$\Rightarrow (\lambda s. Q[\mathsf{fst}\, s\texttt{'A1'}, \ldots \mathsf{snd}\, s\texttt{'Bn'}])s$$

This is precisely as required: $\vdash \forall\, s.\, [\![P]\!]\, s\ \wedge \neg([\![b]\!]\, s) \Rightarrow [\![Q]\!]\, s$.

Underlying the use of the `IF` rule there is therefore a mechanism to translate between theorems of higher-order logic and theorems of the embedded program logic.

$$\texttt{ASS}\ \frac{}{\vdash \{P[\tilde{e}/\tilde{x}]\}\ \tilde{x}\ \texttt{:=}\ \tilde{e}\ \{P\}} \qquad \texttt{SEQ}\ \frac{\vdash \{P\}\ c_1\ \{R\} \qquad \vdash \{R\}\ c_2\ \{Q\}}{\vdash \{P\}\ c_1\ \texttt{;}\ c_2\ \{Q\}}$$

$$\texttt{PRE}\ \frac{\vdash P' \Rightarrow P \qquad \vdash \{P\}\ c\ \{Q\}}{\vdash \{P'\}\ c\ \{Q\}} \qquad \texttt{POST}\ \frac{\vdash \{P\}\ c\ \{Q\} \qquad \vdash Q \Rightarrow Q'}{\vdash \{P\}\ c\ \{Q'\}}$$

$$\texttt{IF}\ \frac{\vdash \{P \wedge b\}\ c\ \{Q\} \qquad \vdash P \wedge \neg b \Rightarrow Q}{\vdash \{P\}\ \texttt{if}\ b\ \texttt{then}\ c\ \{Q\}}$$

$$\texttt{ITE}\ \frac{\vdash \{P \wedge b\}\ c_1\ \{Q\} \qquad \vdash \{P \wedge \neg b\}\ c_2\ \{Q\}}{\vdash \{P\}\ \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{Q\}}$$

Figure 5.2: HOL Derived Rules of Floyd-Hoare Logic

This translation is routine, implemented using higher-order matching, but complicated by the possible appearance of *logical* variables in addition to program variables in the specifications—one is obliged to distinguish between logical variables and program variables by observing some suitable syntactic convention. We note that `IF` is an ML function taking two theorems as arguments, and yielding a theorem as result. ML values of type *theorem* can only be derived in the HOL system by application of the primitive inference rules of higher-order logic—though they can also be introduced as formal axioms. In avoiding axioms, the definitional style of interaction with the theorem prover that we have been following, where constants such as Bind, Spec, *etc.*, are defined in terms of existing constants and logical connectives, ensures that the logic of the HOL system remains consistent. Thus theorems such as $\vdash \{P\}\ c\ \{Q\}$ can be trusted, but whether or not they are meaningful is of course a matter of interpretation.

The other rules of the program logic (see Figure 5.2) are derived in much the same way as `IF`, and uniformly so, from the theorems listed above—all, that is, except for the assignment axiom. In the simple case this must be a function taking terms representing $\{P\}$ and $x\ \texttt{:=}\ e$, and which yields a theorem $\vdash \{P[e/x]\}\ x\ \texttt{:=}\ e\ \{P\}$. Gordon's derivation of the assignment axiom must be generalised in two important respects because of the parallel multi-assignment, and the polymorphism in the type *state*. Thus care has to be taken not only to correctly instantiate the terms in forming $\{P[\tilde{e}/\tilde{x}]\}$, but also their types. Otherwise, the derivation is much as in [35].

5.2.3.2 Tactics for Floyd-Hoare Logic

The inference rules derived above support forward proof in HOL. It is more natural however, when devising proof strategies from scratch, to work backwards in a goal directed manner from the desired theorem. For this tactics are needed which render a goal into a number of (hopefully) simpler goals. Tactics to support reasoning about

partial correctness specifications can be obtained by inverting the corresponding rules of Hoare logic. For example, a combination of the assignment axiom and the rule for strengthening the precondition will yield:

$$\texttt{PRE}\ \frac{\vdash P \Rightarrow Q[\tilde{e}/\tilde{x}] \qquad \texttt{ASS}\ \dfrac{}{\vdash \{Q[\tilde{e}/\tilde{x}]\}\ \tilde{x}\ \texttt{:=}\ \tilde{e}\ \{Q\}}}{\{P\}\ \tilde{x}\ \texttt{:=}\ \tilde{e}\ \{Q\}}$$

Given the goal $? \vdash\ \{P\}\ \tilde{x}\ \texttt{:=}\ \tilde{e}\ \{Q\}$, the appropriate tactic, `ASS_TAC`, will produce as its only subgoal the verification condition $? \vdash\ P \Rightarrow Q[\tilde{e}/\tilde{x}]$. The tactic `IF_TAC` also yields a verification condition as one of its subgoals. These verification conditions may be proven using the full power of higher-order logic, since they are pure logic formulae, though they will be unprovable if the initial goal is not a theorem.

Appropriate combinations of tactics, using *tacticals* for repetition (`REPEAT`) and trial and error (`ORELSE`) will generate the verification conditions for a partial correctness specification $\{P\}\,c\,\{Q\}$ for any command c in the embedded language. This tactic, which Gordon called `VC_TAC`, may be implemented thus

```
val VC_TAC = REPEAT
    (SEQ_TAC ORELSE ITE_TAC ORELSE IF_TAC ORELSE ASS_TAC);
```

The tactic `SEQ_TAC` behaves as follows:

$$\frac{? \vdash\ \{P\}\,c_1\ \texttt{;}\ c_2\,\{Q\}}{? \vdash\ \{P\}\,c_1\,\{Q\} \quad ? \vdash\ \{Q\}\,c_2\,\{Q\}} \qquad \frac{? \vdash\ \{P\}\,c\ \texttt{;}\ \tilde{x}\ \texttt{:=}\ \tilde{e}\,\{Q\}}{? \vdash\ \{P\}\,c\,\{Q[\tilde{e}/\tilde{x}]\}}$$

`SEQ_TAC` first tries to apply the second form, matching the last of a sequence of commands against Ass; if this fails (the match can only be a conditional if sequences of assignments are merged and `skip` commands have been dropped) the heuristic inserts a $\{Q\}$ before the conditional.

These tactics are an essential component in our approach to automating the analysis of Geographic Data invariants. Now we need a theory of those invariants.

5.3 A Theory of Geographic Data Invariants

In the preceding section a state was modelled by a pair of polymorphic functions of type *string* $\rightarrow \alpha$ and *string* $\rightarrow \beta$. It is straightforward, though tedious, to extend this theory to several more state components. Here we shall just model sub-routes, routes and track circuits in the second component as entities of type *bool*, and points in the first component (these having more complex type).

Sub-routes have two states, locked and free, and are thus properly modelled as Boolean variables. Routes have three states: unavailable, available and set, and available and unset. In representing routes by Boolean variables it will only be possible to

verify properties of the set/unset bit and not the 'availability' bit which the data can examine but never modify. The availability bit is an override used exceptionally to temporarily bar a route—*e.g.*, for maintenance purposes. Treating routes as Boolean variables is to assume that a test on the availability (bit) of a route is always passed.

Track circuits also have three states: occupied, clear and undefined. The occupied and clear states are represented in different bits in the track circuit memory: if neither is set the track circuit state is undefined. According to the informal description of the semantics in [9], whenever the occupied bit is set by a data command, the clear bit will be cleared, and vice versa. (Points too have this inversion property—see Section 2.4.2, and Section 5.3.2 below.) In modelling track circuits in a single Boolean variable we therefore assume that their state is never undefined.

The sections that follow develop the HOL theory needed to represent the properties of Geographic Data introduced in Section 3.3.1 and 3.3.3. These are:

MX the mutual exclusion property for sub-routes over a track section, and discussed in Section 5.3.1 below;

PT the points property relating the orientation of the points with the sub-routes through them (Section 5.3.2); and

RT the property relating routes and their component sub-routes (Section 5.3.3).

We shall therefore define the corresponding functions in higher-order logic.

5.3.1 Track Circuits – MX

In higher-order logic the free (`f`) state of a sub-route will be represented by the constant T, and the locked (`l`) state by F, these being the distinct elements of the type *bool*. We shall freely write `f` and `l` instead, whenever this clarifies the the explanation. A plain bidirectional track section will be associated with a pair of sub-routes (a and b, say), thus the mutual exclusion property can be expressed in the HOL term $a \vee b$, or, more verbosely in $a = \mathsf{T} \vee b = \mathsf{T}$. In Section 3.3.3 the *macro* **MX** operated on a list of sub-routes—this can be expressed in HOL by the constant MX, defined using the auxiliary function M:

$$\vdash (\forall s.\mathsf{M}\, s\, [\,] = \mathsf{T}) \;\wedge\; (\forall s\, t\, h.\mathsf{M}\, s\, [h \mid t] = (s \vee h) \wedge (\mathsf{M}\, s\, t))$$

$$\vdash (\mathsf{MX}\, [\,] = \mathsf{T}) \;\wedge\; (\forall s\, t.\mathsf{MX}\, [h \mid t] = (\mathsf{M}\, s\, t) \wedge (\mathsf{MX}\, t))$$

In the sequel two common special cases will arise from which other varieties can be assembled as desired:

$$\vdash \forall a\, b.\mathsf{MX2}(a, b) = (a \vee b)$$

$$\vdash \forall a\, b\, c\, d.\mathsf{MX4}(a, b, c, d) = \mathsf{MX}\, [a, b, c, d]$$

The latter involves $\binom{4}{2} = 6$ terms like MX2, *viz.*:

$$(a \vee b) \wedge (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d) \wedge (c \vee d)$$

This combinatorial explosion of terms is problematic only if the definition is expanded in proofs, but this is never necessary in practice. Apropos MX4, for example, we can prove some simple but essential theorems

$$\vdash \mathsf{MX4}(a, \mathsf{T}, \mathsf{T}, \mathsf{T}) = \mathsf{T}$$

$$\vdash \mathsf{MX4}(a, b, c, d) \Rightarrow \mathsf{MX4}(\mathsf{T}, b, c, d)$$

and so on. The first of these will be useful in simplifying (rewriting) terms in the invariance proof, while the second is a resolution theorem by which fresh hypotheses can be generated from those already given. A collection of such theorems is needed to cover the possible cases since the HOL proof system is not readily able to exploit even first-order unification (although Slind [86] has illustrated that AC unification can be added to the HOL system's proof infrastructure).

To illustrate the use of the above definitions, consider the track circuit T_2 which has two sub-routes T_2^{ab} and T_2^{ba}. The *property* $\mathbf{MX}[T_2^{ab}, T_2^{ba}]$ will be expressed in the higher-order logic term $\mathsf{MX2}(T_2^{ab}, T_2^{ba})$. In the embedded logic this will be expressed in the predicate $\lambda s.\mathsf{MX2}(\mathsf{snd}\, s\,\texttt{'T2AB'}, \mathsf{snd}\, s\,\texttt{'T2BA'})$. If this is the only term in the invariant, and we wish to prove that the sub-route release rule for T_2^{ba} is correct with respect to this, then the goal

$$?\vdash \{\mathbf{MX}[T_2^{ab}, T_2^{ba}]\}\ \texttt{if}\ T_2 = \texttt{c} \wedge T_1^{ca} = \texttt{f}\ \texttt{then}\ T_2^{ba}\ \texttt{:=}\ \texttt{f}\ \{\mathbf{MX}[T_2^{ab}, T_2^{ba}]\}$$

(which is in any case true by Lemma 4.3 on page 85) is reduced by `VC_TAC` and some other machinery to:

$$[\mathsf{MX2}(T_2^{ab}, T_2^{ba})] \vdash \mathsf{MX2}(T_2^{ab}, \mathsf{T})$$

This can be solved immediately using the pre-proved theorems about MX2.

5.3.2 Points – PT

Points are represented by an eight-bit record which is subdivided into two four-bit fields, one holding data for the points normal, and the other reverse. In a data test or command on points it is always necessary to select the normal or reverse field (*e.g.*, P_1 `cn`, or P_1 `crf`). Thus points may be modelled using two four-bit words having the format:

$$\underbrace{[\texttt{c}, \texttt{d}, \texttt{k}, _]}_{\text{normal}}, \underbrace{[\texttt{c}, \texttt{d}, \texttt{k}, _]}_{\text{reverse}}$$

The fourth bit in each field has been masked out here as these are override flags which the Geographic Data can neither write to nor, in this case, read. These flags may be

cleared to temporarily disable the points in one position or the other directly from the technician's console: the control interpreter examines these in evaluating a points "free to move" condition, and only if they are set is the condition passed. Moreover, whenever data are processed which set the normal command bit (`c`), the SSI automatically clears the reverse command bit, and vise versa. The same is true of the detection bits (`d`), updated in processing the incoming status telegrams from the points modules in the railway.

One way to model this behaviour is to suppose, as in the earlier chapters, that points have but a single control variable which can be in one of the two states `cn` or `cr`. Nevertheless, the SSI is still in a well defined state when both control bits are unset (though these circumstances only prevail at startup), and so both should be modelled. This also entails modelling the inversion of the reverse control bit (say) whenever the normal control bit is set/cleared. But in the semantic framework sketched in Section 5.2 this aspect of the 'behaviour' of points was not considered. It is possible to express this inversion of the control (`c`, and detection `d`) bits directly in the semantics of the embedded language, but this entails introducing to the theory of the object language, object level datatypes—product types in particular—and much semantic complexity which it has hitherto been sought to avoid. As far as is possible we prefer to separate the theory of the semantics of GDL from the theory needed to represent the datatypes for points, track circuits, *etc.*.

Let us therefore introduce the abbreviation

$$\mathit{points} = \mathit{bool\ list} \times \mathit{bool\ list}$$

This is a product type in higher-order logic, and the first of the pair is a list of Boolean variables to represent the normal bits in points memory, the second represents the reverse bits. Ideally one would use a package such as the *word* or *record* libraries [100, 98] for HOL which automate the derivation of the representation theorem and access functions for manipulating user declared types such as *points*—but neither was available for the version of the HOL system being used at the time. In any case, we spell out a few of the details that use of the *record* library conceals in defining constants CN and CR to select the appropriate fields from a points record, and the mutator functions for these fields:

$$\vdash \forall p{:}\,\mathit{points}.\mathsf{CN}\,p = \mathsf{hd}\,(\mathsf{fst}\,p)$$

$$\vdash \forall p{:}\,\mathit{points}.\mathsf{setCN}\,p = ([\mathsf{T} \mid \mathsf{tl}\,(\mathsf{fst}\,p)], [\mathsf{F} \mid \mathsf{tl}\,(\mathsf{snd}\,p)])$$

$$\vdash \forall p{:}\,\mathit{points}.\mathsf{CR}\,p = \mathsf{hd}\,(\mathsf{snd}\,p)$$

$$\vdash \forall p{:}\,\mathit{points}.\mathsf{setCR}\,p = ([\mathsf{F} \mid \mathsf{tl}\,(\mathsf{fst}\,p)], [\mathsf{T} \mid \mathsf{tl}\,(\mathsf{snd}\,p)])$$

The `d` and `k` fields are treated similarly. Some easy theorems follow from these defini-

tions which are useful in simplifying goals:

$$\vdash \mathsf{CN}\,(\mathsf{setCN}\,p) = \mathsf{T}$$

$$\vdash \neg(\mathsf{CN}\,(\mathsf{setCR}\,p)) = \mathsf{T}$$

Thus, in order to make matters concrete, the points command P_1 `cn` will be represented in higher-order logic by the term:

$$\mathsf{Ass}([\texttt{'P1'}], [\lambda s.\mathsf{setCN}(\mathsf{fst}\, s\, \texttt{'P1'})], [\,], [\,])$$

This harsh HOL syntax may be rendered invisible by a parser and unparser—albeit one of some sophistication—but we aim of course for batch automation.

Finally, as in Section 3.3.3, the macros $\mathbf{PT}_{\texttt{cn}}$ and $\mathbf{PT}_{\texttt{cr}}$ can be defined, but here in the contrapositive:

$$\vdash (\mathsf{PC}\,([\,]{:}\,\textit{bool list}) = \mathsf{T}) \;\wedge\; (\forall\, h\, t.\mathsf{PC}\,[h \mid t] = (h \wedge \mathsf{PC}\, t))$$

$$\vdash \forall\, p\, s.\mathsf{PNT}\,(p, s) = (\mathsf{CN}\, p) \Rightarrow \mathsf{PC}\, s$$

$$\vdash \forall\, p\, s.\mathsf{PRT}\,(p, s) = (\mathsf{CR}\, p) \Rightarrow \mathsf{PC}\, s$$

The first parameter is a points variable (*e.g.*, P_1) and the second is intended to be a list of sub-routes through those points *in the other orientation* (*i.e.*, $[T_1^{cb}, T_1^{bc}]$, in this case). Recalling Section 3.3, this captures the property that if the points are controlled normal (say) then the reverse sub-routes should be free. From these definitions it easily follows, for example, that $\forall\, p\, s.\ \mathsf{PNT}\,((\mathsf{setCR}\, p)\,, s)$.

5.3.3 Routes – RT

In Section 4.4.3 it was shown that the original, rather simple formulation of **RT** was inadequate. The intuition though is clear: when the route r is set, its sub-routes a, b and c should be locked:

$$(r = \texttt{s}) \Rightarrow (a = \texttt{l} \wedge b = \texttt{l} \wedge c = \texttt{l}) \tag{i}$$

Given that the route is defined by the sub-routes a, b and c *in that order*, we may consider an alternative that captures the same intuition

$$r \Rightarrow a \;\wedge\; a \Rightarrow b \;\wedge\; b \Rightarrow c \tag{ii}$$

(now dropping the equational format, and giving $\Rightarrow$ higher precedence than $\wedge$). This is a stronger property expressing "if the first sub-route is locked then the rest of the route remains locked", and so on. The drawback is that we require to specify the sub-routes along the route in order, (i) being insensitive to order. Nevertheless, using (ii), in contrapositive form, $a = b = \texttt{f}$ can be inferred given $c = \texttt{f}$. It was the unknown

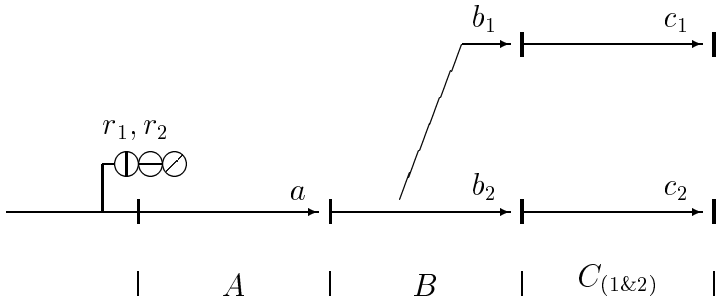


Figure 5.3: Routes that diverge after a common segment

state of b, the intermediate sub-route, that posed the problem in Section 4.4.3. Now if $r \Rightarrow a \ \wedge \ a \Rightarrow b \ \wedge \ b \Rightarrow c$ is invariant, then so is

$$r \Rightarrow (r \Rightarrow a \ \wedge \ a \Rightarrow b \ \wedge \ b \Rightarrow c) \tag{iii}$$

which is equivalent to $r \Rightarrow a \wedge b \wedge c$, the equivalence being due to the following:

Proposition 5.1 For all $n \geq 1$, $r \Rightarrow (r \Rightarrow a_1 \ \wedge \ a_1 \Rightarrow a_2 \ \wedge \cdots \wedge \ a_{n-1} \Rightarrow a_n)$ if and only if $r \Rightarrow (a_1 \wedge a_2 \wedge \cdots \wedge a_n)$. □

Proof By a simple induction on the 'length of the route' n, for example. ■

Upon reflection however (ii) is not correct under all circumstances. To see why consider the two routes in Figure 5.3 which *diverge* at B after following a common route segment over A. Without loss of generality suppose r_1 and r_2 start at the same signal, and a is the first sub-route on each. Generalising (ii) gives

$$\left.\begin{array}{rl} & r_1 \Rightarrow a \ \wedge \ a \Rightarrow b_1 \ \wedge \ b_1 \Rightarrow c_1 \\ \wedge & r_2 \Rightarrow a \ \wedge \ a \Rightarrow b_2 \ \wedge \ b_2 \Rightarrow c_2 \end{array}\right\} \tag{iv}$$

but clearly, in any assignment to the variables satisfying **RT** in which a is locked, b_1 and b_2 must also be locked—yet this contradicts $\mathbf{MX}[b_1, b_2]$ which must hold simultaneously with **RT**. The result is that no invariant designed along these lines can be satisfied by any data which set either of these routes. Naturally this is undesirable (technically, we have eliminated an intended model for **F**). However, the intuition concerning the linkage $A \leftrightarrow B$ is that whenever a is locked, either b_1 *or* b_2 is locked. Instead, (iv) gives $a \Rightarrow b_1 \wedge b_2$, so this is weakened to $a \Rightarrow b_1 \vee b_2$:

$$\left.\begin{array}{rl} & r_1 \Rightarrow a \ \wedge \ a \Rightarrow (b_1 \vee b_2) \ \wedge \ b_1 \Rightarrow c_1 \\ \wedge & r_2 \Rightarrow a \ \wedge \ b_2 \Rightarrow c_2 \end{array}\right\} \tag{v}$$

Although this will be satisfied whenever a, b_1 and b_2 happen to be locked simultaneously, the mutual exclusion property for B, *viz.* $\mathbf{MX}[b_1, b_2]$, ensures that these circumstances are invalidated.

It turns out that (v) can still be improved for it is easy to see that the formula will be satisfied by an assignment to the variables in which $r_2 = \mathtt{s}$ while $a = b_1 = c_1 = \mathtt{l}$. This could have undesirable consequences if one remembers that r_1 may be temporarily barred by the technician's control: if the technician barrs r_1 (the branch line route), but r_1 and r_2 are confused in the Geographic Data for these routes, this would have the effect of barring the mainline route instead. So (v) alone is not enough to guarantee "whenever a route is set, all its sub-routes are locked." There are several ways to strengthen the invariant so as to trap this not unlikely error in the data. The safest is to retain the original formulation given by (i):

$$\left.\begin{array}{ll} & r_1 \Rightarrow a \ \wedge \ a \Rightarrow (b_1 \vee b_2) \ \wedge \ b_1 \Rightarrow c_1 \\ \wedge & r_2 \Rightarrow a \ \wedge \ b_2 \Rightarrow c_2 \\ \wedge & r_1 \Rightarrow (a \wedge b_1 \wedge c_1) \ \wedge \ r_2 \Rightarrow (a \wedge b_2 \wedge c_2) \end{array}\right\} \qquad \text{(vi)}$$

This formula is undeniably complex, increasing the likelihood of introducing specification errors if it has to be defined manually. Fortunately this is unnecessary because the specification

$$\mathbf{RT}(r_1, [a, b_1, c_1]) \wedge \mathbf{RT}(r_2, [a, b_2, c_2]) \qquad \text{(vii)}$$

can be algorithmically massaged to the correct form. In the HOL formulation it is preferable to have (ii) in contrapositive form because this simplifies the mechanisation of the invariance proof in the next section. Define RT thus:

$$\vdash (\forall r.\mathsf{RT}\, r\, [\,] = \mathsf{T}) \ \wedge \ (\forall r\, a\, s.\mathsf{RT}\, r\, [a \mid s] = a \Rightarrow r \wedge (\mathsf{RT}\, a\, s))$$

Then, given a specification in higher-order logic such as (vii), or that on page 60, the definition of RT is expanded everywhere to obtain an equivalent conjunctive formula which is then broken up into the list of its conjuncts (*e.g.*, (iv)). This list (*unsafe*) is then passed to the *Routes* algorithm displayed in Figure 5.4 which generalises the preceding argument. In quadratic time this produces a new list (*keep*) which is then turned back into a conjunctive formula.

Note that the result can be optimised in the inner loop of the program by deleting any duplicated terms in *unsafe*. The formula obtained from *keep* is then strengthened with clauses such as those introduced at (vi), one for each diverging route (requiring an $\mathcal{R}^2$ algorithm, if $\mathcal{R}$ is the number of routes). Once more appealing to the contrapositive define:

$$\vdash (\mathsf{RC}\,[\,] = \mathsf{F}) \ \wedge \ (\forall h\, t.\mathsf{RC}\,[h \mid t] = h \vee (\mathsf{RC}\, t))$$

$$\vdash \forall r\, s.\mathsf{RT1}\, r\, s = \mathsf{RC}\, s \Rightarrow r$$

The result from *Routes* can be further optimised to minimise the number of terms in the invariant by rewriting $b \Rightarrow a_1 \wedge b \Rightarrow a_2$ as $b \Rightarrow (a_1 \wedge a_2)$, *etc.*.

initialise: *keep* $\leftarrow [\,]$
while *unsafe* $\neq [\,]$ do
    let *conj* be head and *rest* be tail of *safe* $= [conj \mid rest]$
    let *a* be consequent and *b* be antecedent of $conj = b \Rightarrow a$
    for $conj' = b' \Rightarrow a'$ in *rest* do
        if $a' = a$ then
            delete $conj'$ from *rest* and
            update *b* so $b \leftarrow b \wedge b'$
    done
    update *keep* so $keep \leftarrow [b \Rightarrow a \mid keep]$
    update *unsafe* so $unsafe \leftarrow rest$
done

Figure 5.4: *Routes*: Massaging the **RT** invariant

Note, finally, that the elaboration in the formal definition of **RT** introduced above is only needed where routes diverge after following a common segment of track. Where routes only converge (like R_2 and R_4) or diverge due to points in the *first* track section (*cf.* R_{02} and R_{04}), (ii) suffices by Proposition 5.1. In these circumstances *Routes* returns a formula logically equivalent to its input.

5.4 Mechanising the Invariance Proof

Turning now to the Geographic Data for WEST (see Appendix C), there are ten panel route requests and nineteen sub-route release rules, there being no data for the inward sub-routes at the fringes of the Interlocking area. The invariant for this system was defined in Figure 3.5, on page 60. Note that there are diverging routes here, namely R_{51} and R_{53}, hence two additional terms will be appended to the formula when the inconsistencies these routes introduce are removed. *Routes* returns the term:

$$
\begin{array}{llllll}
T_1^{ca} \Rightarrow R_{02} & \wedge & T_2^{ba} \Rightarrow T_1^{ca} & \wedge & T_1^{cb} \Rightarrow R_{04} & \wedge \\
T_3^{ba} \Rightarrow T_1^{cb} & \wedge & T_1^{ac} \Rightarrow R_1 & \wedge & T_0^{ab} \Rightarrow (T_1^{ac} \wedge T_1^{bc}) & \wedge \\
T_1^{bc} \Rightarrow R_3 & \wedge & T_6^{ab} \Rightarrow R_5 & \wedge & T_5^{ab} \Rightarrow T_6^{ab} & \wedge \\
T_6^{ba} \Rightarrow R_6 & \wedge & T_7^{ba} \Rightarrow (T_6^{ba} \wedge T_6^{ca}) & \wedge & T_4^{ca} \Rightarrow R_2 & \wedge \\
T_6^{ca} \Rightarrow (T_4^{ca} \wedge T_4^{ba}) & \wedge & T_4^{ba} \Rightarrow R_4 & \wedge & T_6^{ac} \Rightarrow (R_{51} \wedge R_{53}) & \wedge \\
(T_4^{ac} \wedge T_4^{ab}) \Rightarrow T_6^{ac} & \wedge & T_2^{ab} \Rightarrow T_4^{ac} & \wedge & T_3^{ab} \Rightarrow T_4^{ab} &
\end{array}
$$

Following the discussion in Section 4.4, two proof schemas are proposed which will be referred to as `SRR_TAC` (for sub-route release tactic) and `PRR_TAC` (for panel route request tactic) in the sequel. Since the data for WEST are very regular a single tactic could deal with both classes of data—but in general several tactics will be needed. The

verification conditions SRR_TAC and PRR_TAC must solve will have the general form

$$? \vdash (F_1 \wedge F_2 \wedge \cdots \wedge F_n) \wedge b \Rightarrow (F_1 \wedge F_2 \wedge \cdots \wedge F_n)[\tilde{v}/\tilde{x}] \qquad (*)$$

where the F_i are components of the safety property **F**. In the following two subsections we look in detail at how to solve these verification conditions as they arise in the route request and sub-route release data. Section 5.4.3 considers the situations where the tactics developed below do not succeed.

5.4.1 Sub-route Release Data Tactic

Lemma 4.3 on page 85 is given as a metatheorem, stating that the invariant is preserved when clearing a sub-route for appropriate formulations of **MX** and **PT**. Pragmatically, this offers a potential shortcut in implementing SRR_TAC since much of the extraneous computational complexity encountered stems from manipulating long lists of assumptions in the sequents. Unhappily, we cannot prove Lemma 4.3 in the present framework as the semantic embedding is too shallow: there is no metatheory concerning the representation of GDL nor the invariants in higher-order logic on which to draw. In principle, this deficiency can be redressed by providing a deeper semantic embedding, but that would require a radically different framework to that sketched above. For the present the inefficiencies the weaker theory entails can be tolerated.

Proceeding from $? \vdash \{\mathbf{F}\}$ `if` b `then` $\tilde{x}$ `:=` $\tilde{v}\,\{\mathbf{F}\}$ we begin by deriving the verification conditions using VC_TAC. This produces two subgoals from the guarded command; the first of these is briefly postponed using ALL_TAC, and the latter can be solved immediately as it is an instance of the tautology $x \wedge y \Rightarrow x$:

```
fun POP_ONE th =
    ASSUME_TAC th THEN UNDISCH_TAC(concl th) THEN PIMP th;

fun RECURSE () = POP_ASSUM (fn th => STRIP_TAC THENL
    [POP_ONE th,
     RECURSE () ORELSE POP_ASSUM (fn th => POP_ONE th)]);

val SRR_TAC =
         VC_TAC THENL [ALL_TAC, MATCH_ACCEPT_TAC FB_IMP_F]
    THEN RSTRIP_TAC THEN RECURSE ();
```

The remaining goal is of the form $(*)$ and is solved by stripping the antecedent and forming one goal for each conjunct in what remains. With RSTRIP_TAC we are careful to leave this in the form:

$$[F_1, F_2, \ldots, F_n, b] \vdash (F_1 \wedge F_2 \wedge \cdots \wedge F_n)[\tilde{v}/\tilde{x}]$$

The recursive tactic RECURSE implements the finer details of the proof. Inside the loop POP_ASSUM removes (or pops, since the structure holding the assumptions is a stack)

the first assumption from the goal; this is followed by `STRIP_TAC` which reduces ($\wedge$-Elimination) the result to two subgoals

$$[F_2, \ldots, F_n, b] \vdash F_1[\tilde{v}/\tilde{x}]$$

$$[F_2, \ldots, F_n, b] \vdash (F_2 \wedge \cdots \wedge F_n)[\tilde{v}/\tilde{x}]$$

the first of which is solved by the procedure `POP_ONE`, while the second is solved in the next iteration of the loop. The recursion 'bottoms out' when (`STRIP_TAC` fails and) the only remaining goal is $[F_n, b] \vdash F_n[\tilde{v}/\tilde{x}]$.

The tactic `POP_ONE` reintroduces the popped assumption to the first goal

$$[F_2, \ldots, F_n, b] \vdash F_1 \Rightarrow F_1[\tilde{v}/\tilde{x}]$$

and invokes a procedure, `PIMP`, that will prove such theorems efficiently given the known structure of Geographic Data invariants. This structure is inferred by `PIMP` from the parameter `th` which is a theorem of the form $\vdash F_i \Rightarrow F_i$. In particular:

- if F_i's leading combinator is MX2, MX4 or PT the goal may be easily solved by rewriting, using basic facts from propositional logic and/or the pre-proved theorems concerning these constructors;
- otherwise the term is derived from encoding **RT** and is therefore of the general form $(f \Rightarrow g) \Rightarrow f' \Rightarrow g'$. These can usually be solved by a combination of rewriting and resolution.

In the latter case only the assumption b, derived from the guard in the command, is needed to complete the proof. To illustrate, recall the sub-route release rule for T_2^{ba} that served as an example in Section 4.4: T_2^{ba} `f if` $T_2 =$ `c` $, T_1^{ca} =$ `f \ ..` At some point during its traversal of **F**, `RECURSE` generates the subgoal

$$[H] \vdash (\mathsf{T} \Rightarrow T_1^{ca}) \Rightarrow T_2^{ba} \Rightarrow T_1^{ca}$$

and foremost amongst the hypotheses H is the assumption $T_1^{ca}(= \mathsf{T})$ derived from the rule itself. With this fact we can simplify the goal and, since $t \Rightarrow \mathsf{T}$ for any t, eventually conclude that the sub-route release rule for T_2^{ba} satisfies the invariant. However, when the only useful fact at our disposal is $T_1^{ac}(= \mathsf{T})$, as is the case with the erroneous version of this rule also considered in Section 4.4, this step in the proof fails so the tactic as a whole fails.

5.4.2 Route Request Data Tactic

The finesse with which `SRR_TAC` was approached is not to be repeated in defining the route request data tactic. `PRR_TAC` essentially solves its goal by a brute force rewriting strategy:

```
val BTHEN_TAC =
         REWRITE_TAC[PT,PC,RT1,RC]
    THEN STRIP_TAC
    THEN TWICE RES_TAC
    THEN ASM_REWRITE_TAC (PT_THM::MX_THMS);

val PRR_TAC =
         VC_TAC
    THENL [BTHEN_TAC THEN NO_TAC, MATCH_ACCEPT_TAC FB_IMP_F];
```

Since the data for WEST are one-armed conditionals VC_TAC always reduces the initial goal to two subgoals: as before, the second of these is an instance of a trivial pre-proved theorem, while the former is like $(*)$. BTHEN_TAC solves such goals by first expanding the definitions of PT (and RT1 if needed) and stripping ($\Rightarrow$-Elimination) the antecedent of this implicative goal:

$$[b, F_n, \ldots, F_1] \vdash (F_1 \wedge F_2 \wedge \cdots \wedge F_n)[\tilde{v}/\tilde{x}]$$

At the next step in the proof there is much scope for refining the tactic. TWICE is a tactical that applies its tactic argument to the goal *twice*. In this case RES_TAC is used to search (exhaustively) among the hypotheses for assumptions a and $a \Rightarrow b$, by which b may added. Assumptions like b and $a \wedge b \Rightarrow c$ work equally well to add $a \Rightarrow c$ to the hypotheses. This resolution step is applied twice (here) because a further pass is needed to deduce c given a.

For a concrete example take the proof required for *Q2. Due to the diverging routes R_{51} and R_{53}, the term $T_4^{ab} \wedge T_4^{ac} \Rightarrow T_6^{ac}$ appears amongst the hypotheses in the goal after applying STRIP_TAC. By the guard in the command $T_4^{ac} = \mathsf{T}$; furthermore, P_2 crf is also a guard, so from PRT$(P_2, [T_4^{ab}, T_4^{ba}])$ one can deduce that these *normal* sub-routes over T_4 are free. However, two applications of RES_TAC are needed to deduce T_6^{ac} from the hypotheses T_4^{ac} and T_4^{ab}. Following the discussion in Section 4.4.3 this intermediate result is needed to prove **MX** invariant.

The final step in PRR_TAC is to rewrite the goal using all appropriate assumptions, some primitive (built-in) facts about propositional logic and the pre-proved theorems about PRT, MX2 and MX4, *etc.*. ASM_REWRITE_TAC will replace any terms in the goal which also appear amongst the hypotheses by T, and similarly any terms that are instances of the supplied and built-in theorems. HOL's rewrite engine applies the rewrite theorems repeatedly until either the goal reduces to T, in which case PRR_TAC succeeds, or until no further change occurs in the goal (in which case PRR_TAC fails since there remain unproved subgoals and NO_TAC is a tactic that always fails).

The success of this procedure therefore depends upon the resolution step yielding all the necessary intermediate results. Generalising the tactical TWICE, the tactic BTHEN_TAC can be implemented thus:

```
fun BTHEN_TAC n = ...
    THEN (repeat n) RES_TAC THEN ...
```

The integer n is supplied by the program calling this tactic. It turns out that (in most cases) n need be no more than the length of the route—which can be ascertained by a purely syntactic check on the panel route request rule to which the tactic is being applied. Exceptional cases may arise if opposing routes bifurcate more than once. However, it is perhaps ultimately more satisfactory to implement `RES_TAC`, one of the theorem prover's primitive tactics, in such a way as to reapply itself as long as fresh hypotheses are being discovered (this can be simulated using the tactical `CHANGED_TAC`, but is it not very efficient). Note that we should not attempt to use

```
val BTHEN_TAC = ...
    THEN REPEAT RES_TAC THEN ASM_REWRITE_TAC ...
```

since this would diverge whenever the combination of resolution and rewriting fails to solve the goal completely (since neither tactic fails).

Computationally, while resolution is itself rather expensive, most of the complexity inherent in this tactic (or to the proof it performs) is introduced by the innocent-looking `STRIP_TAC`. The reason for this is that when there is a disjunctive term in the antecedent of the goal $(*)$, `STRIP_TAC` yields several subgoals:

$$\frac{? \vdash\ a \vee b \Rightarrow c}{[a] \vdash\ c \quad [b] \vdash\ c}$$

This will be the case whenever a 'points controlled or free to move' test is included in the availability conditions for a route (*e.g.*, P_3 `cnf` in `*Q51`). Thus, if there are p such tests (say), there are 2^p very similar subgoals in the proof. Controlling the rewriting procedure and the efficiency of the resolution step therefore offer the main opportunities for improving the computational complexity of `PRR_TAC` (but see Section 7.2).

5.4.3 Failed Tactics

The failure of a tactic on the goal $? \vdash \{\mathbf{F}\}\, c\, \{\mathbf{F}\}$ does not imply the data are incorrect—the tactics discussed above are not complete proof procedures. From the logical standpoint there are three situations that should be borne in mind when the tactic intended to prove the goal fails:

1. The purported theorem is simply not true because the data violate the safety property **F**. In the case of the erroneous rule for T_2^{ba} mentioned above

   T_2^{ba} `f if` $T_2 =$ `c` , $T_1^{ac} =$ `f \ .`

   a visual inspection of the scheme plan is enough to identify the error. Likewise, the error in the route request rule

   `*Q51 if` P_3 `cnf` , P_2 `cnf` , T_6^{ca} `f` , T_2^{ba} `f`
   `then` R_{51} `s` , P_3 `cn` , P_2 `cn` , T_6^{ac} `l` , T_4^{ac} `l` , T_2^{ab} `l \ .`

will be so revealed, but with perhaps a little more difficulty. Our verification strategy is designed to isolate exactly these sorts of common data errors.

2. The data are correct but the heuristic implemented by the tactic is not sufficiently powerful. This is very likely when one is devising the tactics in the first place, and it is inevitable that some experimentation is needed before one's tactics are sufficiently robust (this was one of the reasons for using HOL after all). Thus, as it stands, `SRR_TAC` will likely need to be reengineered if the invariant is radically changed—instantiating a new dataset will not be a problem, but covering safety properties other than **MX**, **PT**, and **RT** may be. Likewise, since real *PRR* data are often more complex than the guarded commands used to exercise the GDL theorem prover, this tactic too will have to be reengineered to accommodate a wider class of panel request data. An example to illustrate both issues will be given in Section 7.2 where we examine a live set of Geographic Data, and the problem of *overlaps* in particular.

3. Due to Gödel's Incompleteness Theorem there are theorems that are expressible in higher-order logic but which are not provable within its proof system. Thus $\vdash \{\mathbf{F}\}\, c\, \{\mathbf{F}\}$ may be true but unprovable. Nevertheless, the *verification condition* is expressed as a conjecture in the propositional, first-order fragment of higher-order logic, and this is decidable. Completeness of the underlying assertion language is a necessary condition, though in general not sufficient, for the completeness of Floyd-Hoare logic. By restricting to a decidable assertion language we should be able to prove the verification conditions when they are true.

Another source of 'failure' warrants mention. The theorem $\vdash \{\mathbf{F}\}\, c\, \{\mathbf{F}\}$ attests only that c leaves **F** invariant. According to the definition **F** is insensitive to the error in: T_2^{ba} `f if` $T_1 =$ `c` $, T_1^{ca} =$ `f \ ..` As pointed out in Chapter 3, many such errors may be identified by a purely syntactic analysis: this is therefore an important precursor to the behavioural analysis undertaken with the theorem prover.

To summarise this section therefore, two simple proof methods have been implemented in HOL tactics to discharge the verification conditions arising in the safety analysis of sub-route release, and route request data. In the case of `SRR_TAC` the tactic is as efficient as possible since it is clearly linear in the size of **F**, and there is seldom (if ever) a need to resolve amongst the hypothesis to verify the *FOP* data. The same is not true of `PRR_TAC` where resolution is the key to discharging these proofs automatically and, moreover, where the proofs are complicated by disjunctive terms in the antecedent due to points tests in the *PRR* data. Complexity issues are discussed further in the next section.

5.5 Decomposing Global Invariance

In Section 3.5 we were able by dint of an exhaustive proof strategy to verify some safety properties of a few small collections of Geographic Data. The same data (and Sun workstation) were used to exercise the HOL proof method, the results being displayed in Figure 5.5. The first observation is that for an Interlocking as trivial as WEST the direct enumeration approach is by far the quicker! Nevertheless, the algorithmic complexity of the HOL approach is a considerable improvement. In the table below u is the number of sub-route release rules and r is a measure of the number of route request rules taking into account the doubling caused by points tests.

INTERLOCKING	$(\mathcal{P},\mathcal{R},\mathcal{U})$	$(\mathcal{T})$	u	r	Time *FOP*	Time *PRR*
WEST	(3,10,22)	(8)	19	28	175 s	210 s
EAST-WEST	(4,14,32)	(12)	28	34	415 s	515 s
FOREST LOOP	(4,16,32)	(12)	30	32	435 s	515 s
THORNTON JN.	(6,16,40)	(14)	36	56	720 s	850 s

Figure 5.5: Experiments using HOL on some simple Geographic Data

5.5.1 Computational Complexity (Revisited)

It is difficult to give a precise measure of the computational complexity of the proof procedures outlined in the previous section. Clearly there is one proof to conduct for each rule in the data, so as long as the invariant remains unchanged the method has *linear* time complexity. However, as the data and the physical extent of the interlocking grows, so too will the (global) invariant.

For `SRR_TAC` at least, the time complexity is directly proportional to the number of terms in **F**. This, in turn, depends on $\mathcal{P}$, $\mathcal{R}$, $\mathcal{U}$ and $\mathcal{T}$. Assuming that no track section contains more than one set of points $|\mathbf{MX} \wedge \mathbf{PT}| = \mathcal{T} + 2\mathcal{P} \leq 2(\mathcal{T} + \mathcal{P}) = \mathcal{U}$. Less certainty concerns $|\mathbf{RT}|$, but evidently $|\mathbf{RT}| \leq \mathcal{U} + \mathcal{R}$ once the invariant has been rewritten according to the *Routes* algorithm—one conjunct for each sub-route, plus at most one extra for each route. Since $\mathcal{U} \approx u$, the number of sub-route release rules, time complexity for analysing the *FOP* data is quadratic in $\mathcal{U}$.

The empirical evidence indicates that the quadratic measure is overly optimistic. This is because it represents the (theoretically) best attainable time complexity. Coding inefficiencies inherent to the theorem prover account for the discrepancy (for example, in manipulating lists of assumptions and, especially, in handling very large terms). However, for `PRR_TAC` it is more difficult to obtain a useful measure of the complexity. Partly this is because the complexity of the resolution step is difficult to quantify. A

more serious reason is that the number of intermediate proofs that the theorem prover has to perform on each invocation of the tactic grows exponentially with the number of points on the route, but the measure $(2\mathcal{U} + \mathcal{R})\, r$ provides little intuition because r is not a function of the other parameters. (It turns out that the exponential 'in' r can be eliminated, leaving the measure $(2\mathcal{U} + \mathcal{R})\mathcal{R}$, by suitably strengthening the invariant—*i.e.*, adding another safety property. This will be described in Section 7.2.)

Abandoning the search for a global complexity measure, it is nevertheless instructive to consider how the complexity of discharging each verification condition grows with the size of the interlocking under investigation. With the assumption that no track section contains more than one set of points, the number of terms in the invariant $\mathbf{F}$ was noted above: $|\mathbf{F}| \approx 2\mathcal{U} + \mathcal{R}$ (which is a good approximation even without the assumption). The number of *variables* in the term is of course equal to $\mathcal{P} + \mathcal{R} + \mathcal{U}$, and $\mathcal{P}$ and $\mathcal{U}$ are linear functions of $\mathcal{T}$—*i.e.*, adding one track section (circuit) adds *at most* one point switch and four sub-routes. (The number of routes $\mathcal{R}$ does not depend on $\mathcal{T}$, but it is unlikely to exceed the number of sub-routes, $\mathcal{U}$, in practice.)

A natural question is whether one might not redesign the algorithm for discharging the proofs more efficiently. In the next subsection a simple approach to decomposing these global proof steps is proposed, but an alternative is to encode the verification conditions in propositional logic, and call an external decision procedure to check whether $\mathbf{F} \wedge b \Rightarrow \mathbf{F}'$ is falsifiable. Two candidates emerge:

- Binary Decision Diagrams: These were considered in the context of symbolic model checking in Section 3.4.3. The advantage now is that it is not necessary to represent the transition system of the model, which was the problem with model checking. The BDD representation of $\mathbf{F} \wedge b \Rightarrow \mathbf{F}'$ may be efficient in spite of disjunctive components in b because the graph structure representing $\mathbf{F}$ may be shared by all branches. The tautology check is then linear in the size of the BDD, but the size of the graph depends critically on the ordering imposed on the propositional variables involved in the computation.

- NP-TOOLS: The theorem prover marketed by Logikkonsult [88, see Section 1.5] implements remarkably efficient heuristics for natural deduction style proofs in propositional logic. The evidence, supported by Groote *et al.* in a recent application to railway signalling [37], is that this theorem prover is very much more effective at proving 'simple theorems' than are BDD based approaches. Computationally, the efficiency of NP-TOOLS depends on the number of simultaneous (free) assumptions that must be recorded in the natural deduction proof [87]—in practical applications this is usually close to zero. The method is relatively insensitive to the number of propositional variables involved.

In either case, a serious disadvantage of using external decision procedures is that it undermines the logical coherence of the proofs conducted with the HOL system. Arguably this is of little significance since it is plainly the verification conditions themselves that demonstrate 'safety', not the Hoare triples *per se*; but this argument fails to take into account the advantage that can be gained from the compositional nature of Floyd-Hoare logic, or the additional confidence obtained by submitting the proof generated to an independent *proof checker*. This latter option is impossible when external decision procedures are used to complete parts of HOL proofs.

On the other hand, the integration of BBD based verification and deductive theorem proving is an area of active research, although still in its infancy. Harrison [40] describes an interesting experiment along these lines, where he implemented a BDD based tautology checker as a *derived* rule of higher-order logic (thus maintaining the logical coherence of the system, and the security of the proofs). If such tools become part of the mainstream apparatus of the HOL system, the derived decision procedures may well avoid some of the inefficiencies of tactics like `PRR_TAC`.

5.5.2 Heuristics for Decomposition in the Proof

A polynomial time global safety analysis of the data is of significant interest, but practically the implemented proof method is still rather slow. Improving the efficiency of the infrastructure underlying the theorem prover would speed up the process, but this only addresses one cause of the problem. Since **F** is a conjunction of a large number of *local* safety properties it is very likely that an effective decomposition strategy will emerge. To this end, Ingleby [47] has described some of the underlying principles which he derives from Galois theory. It is instructive to relate the main idea here since it explains how to partition the network into non-overlapping segments.

The signalling scheme plan and Geographic Data identity files define an incidence relation—a binary relation $I \subseteq A \times C$ between physical track *attributes* such as points and track circuits, and logical *control* elements such as routes and sub-routes over those attributes. This induces a Galois connection: essentially, given a set of tracks and points $F \subseteq A$, the Galois connection gives a set $F^{\perp} = \{c \in C \mid \neg\exists a \in F \,.\, (a, c) \in I\}$ which is the set of routes and sub-routes that *do not* pass over any of the elements in F. For $G \subseteq C$, $G^{\perp}$ is dually defined. Ingleby searches for sets $X \subseteq C$ that are closed in the sense that $X = X^{\perp\perp}$. The intuition is that a closed set of control elements X is maximal with respect to sharing of attributes in A.

Given closed $X \subseteq C$, if the set $A - X^{\perp}$ is also closed the pair define what Ingleby calls a *locality*. For example, the pairs:

$$(\{T_0, T_1, T_2, T_3, P_1\}, \{R_{02}, R_{04}, R_1, R_3\} \cup \{\text{sub-routes over } \{T_0, T_1, T_2, T_3\}\})$$
$$(\{T_4, T_5, T_6, T_7, P_2, P_3\}, \{R_2, R_4, R_5, R_6\} \cup \{\text{sub-routes over } \{T_4, T_5, T_6, T_7\}\})$$

are (in terms of *attributes*) mutually disjoint localities in WEST. Note, in passing, that the long routes R_{51} and R_{53} are in neither of these partitions. Such relationships are easy to derive automatically from the Geographic Data. These are used by Ingleby and Mitchell [48] to guide heuristics that limit the search space in proving safety by enumeration of the states of the automaton.

The same Galois theory may be adapted to our HOL setting to guide the overall analysis, but this has not been investigated thoroughly. Instead, some rather obvious heuristics are described below for decomposing proofs like $\vdash \{P \wedge Q\}\, c\, \{P \wedge Q\}$. We also *partition* the network structure, as above, keeping routes whole. The observation is that the truth of $\{Q\}\, c\, \{Q\}$ (say) ought to be computationally trivial when execution of the command c has no bearing on the truth of Q.

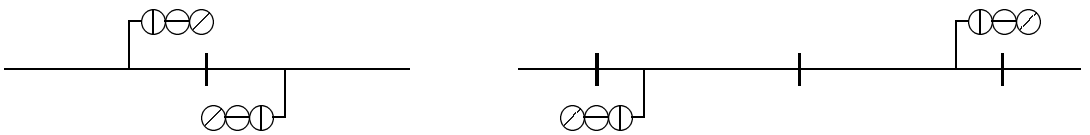
More formally, given a set of assumptions $\mathcal{A}$ and conjunctive goal $\mathcal{I}$, it is convenient to break up the proof of $\mathcal{A} \vdash \mathcal{I}$ into smaller proofs: $\mathcal{A} \vdash \mathcal{I}_1 \ldots \mathcal{A} \vdash \mathcal{I}_n$. Then a derived rule of higher-order logic can assemble these into the desired theorem. `SRR_TAC` works in essentially this manner:

$$\begin{array}{ll} & a_1 \wedge \cdots \wedge a_n \wedge b \Rightarrow a'_1 \wedge \cdots \wedge a'_n \\ \Leftrightarrow & (a_1 \wedge \cdots \wedge a_n \wedge b \Rightarrow a'_1) \wedge \cdots \wedge (a_1 \wedge \cdots \wedge a_n \wedge b \Rightarrow a'_n) \\ \Leftarrow & (a_1 \wedge b \Rightarrow a'_1) \wedge \cdots \wedge (a_n \wedge b \Rightarrow a'_n) \end{array}$$

Here $a_i \wedge b$ is (usually) sufficient for a'_i when the data are correct. In general, more powerful heuristics for the proof's decomposition are needed. An heuristic $\mathcal{H}$ will select subsets of the assumptions: $\mathcal{H}(\mathcal{A}, \mathcal{I}_j) = \mathcal{A}_j \subseteq \mathcal{A}$. Monotonicity of the logic guarantees soundness in that $\mathcal{A}_j \vdash \mathcal{I}_j$ implies $\mathcal{A} \vdash \mathcal{I}_j$.

5.5.3 Static & Dynamic Decomposition

Firstly, givern the theorems $\vdash \{P\}\, c\, \{P\}$ and $\vdash \{Q\}\, c\, \{Q\}$ we can derive the theorem $\vdash \{P \wedge Q\}\, c\, \{P \wedge Q\}$ in our HOL theory. Then, *given* a decomposition like $\mathbf{F} \equiv \mathbf{F}_1 \wedge \mathbf{F}_2$, and a rule c which we suppose to be of the form `if` b `then` $\tilde{x}$ `:=` $\tilde{v}$, the proof proceeds with two separate goals: $? \vdash \{\mathbf{F}_1\}\, c\, \{\mathbf{F}_1\}$ and $? \vdash \{\mathbf{F}_2\}\, c\, \{\mathbf{F}_2\}$. For good choices of $\mathbf{F}_1$ and $\mathbf{F}_2$ the sets of program variables mentioned in each term will have a small intersection. We can then expect to find that for a significant number of the rules c in the database the program variables in c and $\mathbf{F}_2$ (say) will be disjoint. When this is the case the second of these subgoals is trivial since the verification condition $\mathbf{F}_2 \wedge b \Rightarrow \mathbf{F}_2$ matches a simple theorem of propositional logic. This test is a syntactic condition which is sufficient to prove the initial goal. Also, whenever this holds `VC_TAC` generates the above verification condition so the tactic implementing the proof steps need only match the appropriate theorem. However, when the syntactic condition does not hold the tactic must resort to solving $? \vdash \{\mathbf{F}_1 \wedge \mathbf{F}_2\}\, c\, \{\mathbf{F}_1 \wedge \mathbf{F}_2\}$ directly.



(a) (b)

Figure 5.6: A distinction between the network and route structure. In (b) the routes terminating at the signals overlap, in (a) they do not. In (a) the route structure coincides with the network structure, but not the latter case.

The question then is to decide how to separate **F** into its two (or more) components. The Galois theory would help here but we only envisage partitioning the invariant, never the Geographic Data, because the logical route structure and the network structure seldom coincide in practice. This can be seen in the difference between the two parts of Figure 5.6. Given the form of the invariant it is clear that **MX** encourages a decomposition based on the physical (or geographic) structure of the interlocking, while **RT** encourages adherence to the logical structure in 'keeping routes whole'.

In the THORNTON JN. example (see page 216) the notional boundary corresponds to a natural interlocking boundary. Thus, intuitions from signalling engineering are used to decompose the invariant according to the geographic separation. We have sought to minimise the number of routes that straddle the boundary while keeping the partitions evenly proportioned. For the *PRR* data for THORNTON JN. this achieved a 20% improvement in the processing time; on extending THORNTON JN. by adjoining an interlocking similar to WEST, this simple decomposition strategy resulted in a 40% improvement over the corresponding global proof.

In the forgoing analysis it was assumed that the decomposition into $\mathbf{F}_1$ and $\mathbf{F}_2$ was given *a priori*. The other heuristic considered here is similar, but dynamic in character. From the initial goal we can proceed as follows:

$$
\begin{array}{ll}
 & ? \vdash \{\mathbf{F}\}\ \texttt{if}\ b\ \texttt{then}\ \tilde{x}\ \texttt{:=}\ \tilde{v}\ \{\mathbf{F}\} \\
\Leftrightarrow & ? \vdash \mathbf{F}_1 \wedge \mathbf{F}_2 \wedge b \Rightarrow (\mathbf{F}_1 \wedge \mathbf{F}_2)[\tilde{v}/\tilde{x}] \quad \& \quad ? \vdash \mathbf{F} \wedge \neg b \Rightarrow \mathbf{F} \\
\Leftrightarrow & ? \vdash \mathbf{F}_1 \wedge \mathbf{F}_2 \wedge b \Rightarrow \mathbf{F}_1[\tilde{v}/\tilde{x}] \quad \& \quad ? \vdash \mathbf{F}_1 \wedge \mathbf{F}_2 \wedge b \Rightarrow \mathbf{F}_2 \\
\Leftrightarrow & ? \vdash \mathbf{F}_1 \wedge \mathbf{F}_2 \wedge b \Rightarrow \mathbf{F}_1[\tilde{v}/\tilde{x}]
\end{array}
$$

This again supposes that the program variables in $\mathbf{F}_2$ (say) and the command c are disjoint. However, the separation into $\mathbf{F}_1$ and $\mathbf{F}_2$ is not predetermined—the tactic implementing these steps should perform the task differently for each command.

It follows from the tautology $(x \Rightarrow z) \Rightarrow (x \wedge y \Rightarrow z)$ that if $\mathbf{F}_1 \wedge b \Rightarrow \mathbf{F}_1[\tilde{v}/\tilde{x}]$ can be proved, the initial goal can be proved as well. There may be some doubt over whether $\mathbf{F}_1 \wedge b$ is sufficient in general—but when it is not the goal in the last line

above can be tried instead. Note that in order to implement this heuristic we require to rearrange the terms in **F** in a manner governed by the syntactic form of, and the program variables appearing in, the GDL command in the initial goal. For optimal performance modifications to the interface to the theorem prover (the subgoal package which manages the proof) would be needed to provide efficient data structures for holding and manipulating Geographic Data invariants, to improve the performance of rewriting and resolution strategies, *etc.*. The HOL theorem prover being an open ML programming system facilitates such application specific refinements in a natural and logically secure manner.

5.6 Summary

In this chapter an unsophisticated use of the HOL theorem prover has been made to realise a fully automated tool for checking safety properties of Geographic Data. This has been achieved by representing the syntax and semantics of the Geographic Data Language in higher-order logic, and through a semantic embedding of the associated program logic in the HOL system. The theorem prover's tactic language (together with some routine ML programming) have been used to automate proofs of Floyd-Hoare assertions of the form $\{\mathbf{F}\}\, c\, \{\mathbf{F}\}$, for commands c taken from the *PRR* and *FOP* data. There are essentially two circumstances in which the theorem prover fails to prove such conjectures: either the tactic implementing the proof is inadequate, for instance because the form of c introduces subgoals the tactic was not designed to handle (refinement of the tactic is required); or, which is more likely, because the command c does not satisfy the safety criteria encoded in the invariant **F** (indicating a coding error in c or, exceptionally, a specification error in **F**). Being able to pinpoint errors in the data with such precision is a considerable advantage of our proof methodology.

While the prototype GDL verifier described might not be very fast, the potential of the approach has been shown, and its utility demonstrated for interlockings as 'complex' as THORNTON JN.—which is about one third of the scale of a real Interlocking. In Chapter 7 we shall use the theorem prover to tackle the route locking data for the Leamington Spa signalling scheme which represents a typical SSI installation. Success in this venture rests on the compositional nature of Floyd-Hoare logic, the major strength of the approach, and on being able to decompose the global safety proof as per the illustration in Section 5.5 above. Not unnaturally the Leamington Spa case-study raises issues that have not hitherto been ventilated, particularly in generalising the approach discussed above to safety properties of other classes of Geographic Data.

If one wishes to prove properties of GDL programs a precise mathematical statement of the semantics of the language is essential. To formalise, *i.e.*, to *mechanise*,

such proofs a formal statement of the language's semantics is mandatory. This focus on semantics represents a considerable departure from the work of other authors on the subject, particularly Ingleby and Mitchell [48], and Pulley and Conroy [21], where both sets of authors *tacitly* assume that the code used to generate their automata is equivalent, in some sense, to that running in the SSI. In raising the issue of the semantics of the Geographic Data Language at the outset we address the somewhat *ad hoc* character of the language's definition; the novelty of this approach to checking safety properties of the data is to use the semantics of the language to *define* the GDL verifier which we have realised using HOL.

On the face of it, therefore, there is apparently a weakness in the proof methodology put forward in this chapter in that it is not known whether the semantics formalised in HOL are entirely faithful to the semantics of the language as implemented by the SSI control interpreter. However, in Section 5.2 we sought only to demonstrate that the language can be formalised so as to offer the highest degree of automation in checking safety properties of Geographic Data—it is not claimed that every detail is correct. The semantics *are* thought to be valid but there are several questions to address, for instance with respect to points:

- Whether it is admissible to split the datatype into two components?
- Whether the inversion of the control bits is faithfully modelled?
- Whether *PFM* conditions are properly dealt with in the translation?

These are answered in the affirmative in Section 2.4 where the semantics of GDL were first discussed. For the second point, note that two (machine) operations are necessary to set the reverse control bit (say) and clear the normal—no single bit-mask can achieve this in one step and leave the other fields undisturbed. In refining the representation of points, track circuits, *etc.*, in the theorem prover it would be better style to introduce record types and define appropriate selector and mutator operations on the type. HOL libraries are available to automatically derive the abstract characterisation of such datatypes, so we have not dwelt on the details.

The style of semantic embedding given in Section 5.2 is shallow in the sense that one cannot prove 'deep' theorems *about* the semantics of the embedded language. Another approach is suggested in the work Camilleri and Melham [15] who describe HOL utilities to support deeper reasoning about languages such as GDL. Their scheme automates the derivation of the abstract characterisation of inductively defined relations in higher-order logic—a natural candidate here would be the *operational* semantics of a simple programming language like GDL. There is a distinction to note between Gordon's method and that of Camilleri and Melham: in the former case, as in Section 5.2, the axiomatic semantics (Floyd-Hoare logic) are manually derived from the

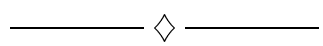
denotational semantics; in the latter, one must first prove that the proposed rules of the program logic are sound with respect to the operational semantics.

Adopting this alternative style of presentation would place the safety analysis of Geographic Data on quite a different, and perhaps slightly richer, mathematical foundation where one could in principle approach the metatheory required to prove simple properties such as Lemma 4.3 which states that clearing a sub-route leaves **MX** invariant. On the other hand it is questionable whether such elaboration is practically necessary: it is only in so far as $\vdash \{\mathbf{F}\}\, c\, \{\mathbf{F}\}$ is a desirable thing in itself. For if, where c is a sub-route release rule, one can be satisfied with a *syntactic* check that these data are properly formed—*e.g.*, that they indeed clear the sub-route to which they refer—one may be perfectly content to prove only the weaker theorem $\vdash \{\mathbf{RT}\}\, c\, \{\mathbf{RT}\}$.

These considerations are not merely stylistic: the efficiency of the proof method is still at issue, and a decomposition of the invariant such as this, enhanced by a deeper semantic framework within which to formulate the invariants we wish to prove, would considerably further the utility of the tool. As demonstrated in Section 5.5, even relatively simple decompositions, based on the rule

$$\frac{\vdash \{P\}\, c\, \{P\} \qquad \vdash \{Q\}\, c\, \{Q\}}{\vdash \{P \wedge Q\}\, c\, \{P \wedge Q\}}$$

were effective in speeding the overall safety analysis. It will be necessary to use further decompositions of this kind to analyse the Leamington Spa data in Chapter 7.



The issue of decomposition in the proof leads to the question of composition in interlocking design. The decompositions favoured earlier followed from the engineering discipline employed to distribute the control of large or complex signalling schemes between a number of cooperating interlockings. The general idea is to 'draw the line through the signal post' since this entails fewer communication overheads: the routes up to the signal are controlled by one interlocking, those forward from the signal by the other. However, where trains can run in both directions over the same track this inevitably leads, as in Figure 5.6(b), to severed routes in one direction or the other. Matters are complicated if there are points in the fringe track section beyond the signal, or if the boundary unavoidably passes through a set of points.

Clearly then, in order to set a route traversing a boundary the adjacent Interlockings need to communicate in some secure manner. Certain data such as the availability of the tail (or remote) portion of a route are required to be communicated only sporadically, but to control signal aspects up to the boundary data such as the status of signals and track circuits in the fringe area are required to be communicated in a continuous basis. These inter-SSI communications, and the safety concerns raised by the remote route locking protocol in particular, are the topic for discussion in the next chapter.

Chapter 6

Distributed Control in Complex Interlockings

In the analysis of the static safety properties of Geographic Data it has not been necessary to attribute any particular behaviour to trains—thus they might appear and disappear at the periphery of the controlled area at will. But clearly this traffic comes from, and goes to, somewhere. In fact the control of large signalling areas will be divided between a number of Interlockings which must act in concert to ensure the overall safety of trains in the network. In particular, two or more Interlockings will need to cooperate to set routes that cross the boundaries between them. The first two sections below recall the main ideas from Section 1.4, and explain how this *remote route locking* is implemented partly in the data, and partly in the generic program. Then, in Sections 6.3 and 6.4, a CCS model is developed and its properties examined. Although the given protocol is found to have certain unsafe features, the formal analysis shows that these can easily be eliminated.

6.1 Introduction

The inter-SSI communications utilise a high speed communications bus called the internal data link (IDL). Several Interlockings can be connected the link, but normally an individual need only exchange data with its nearest neighbours. Although each SSI broadcasts its data, communication is one-to-one rather than one-to-many: each Interlocking connected to the link reads the broadcast data, but extracts only those telegrams that have been addressed to it. Given a cyclic communications strategy, there are two broad classes of data that need to be communicated: continuously required data such as signal aspects used to calculate the aspects of remote signals, and intermittently required data such as are needed to set routes or control points near the boundary.

In the simplest case where a pair of unidirectional lines connect two interlocking areas the boundary can be drawn “through the signal post”. In these circumstances,

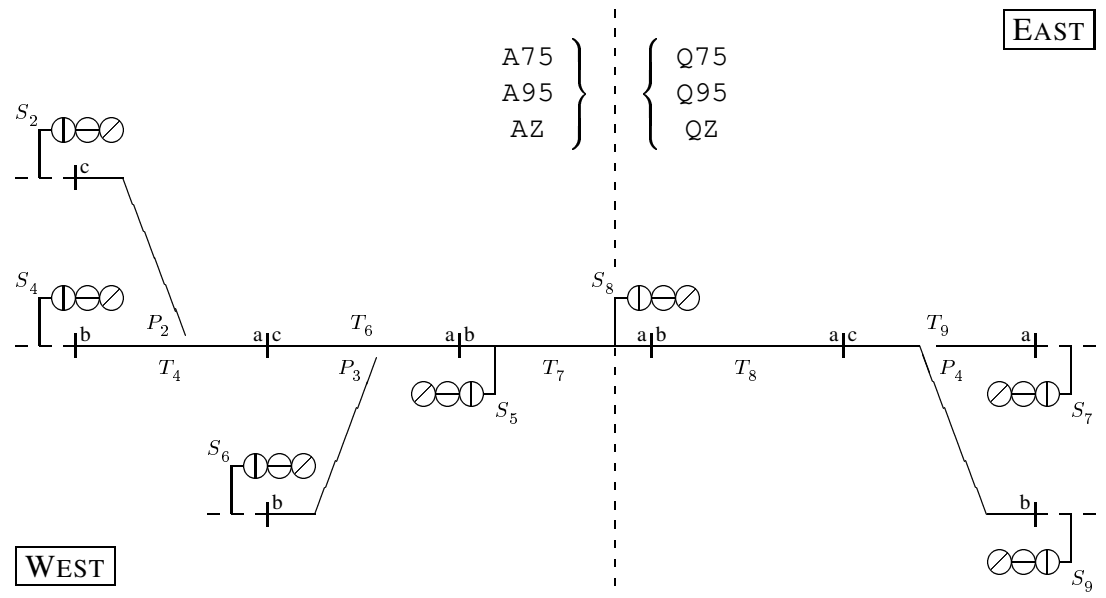


Figure 6.1: EAST-WEST—Setting routes across SSI boundaries

on the outgoing line, the aspects of signals and the status of track circuits beyond the boundary are needed to control signals up to the boundary, and the condition of track circuits up to the boundary will be needed by the other SSI to calculate the aspects of signals beyond the boundary (*cf.* Figure 6.1, where the notional boundary passes through S_8). The IDL simulates the delivery of these data, which would normally arrive over the track-side data highway, typically compressing all of the data required into a single telegram. No route locking over the boundary is necessary in such (ideal) circumstances.

More complicated circumstances arise where route locking across the boundary is necessary, or where the boundary must pass through a set of points. The Interlocking controlling the entrance signal on a cross-boundary route must request the adjacent Interlocking to set that part of the route (the *tail* portion of the route) that is under its jurisdiction. This arises as a consequence of the simple design of the SSI where the TFM inputs are not multiplexed, and there is no shared state (memory) in the distributed control system. Moreover, to control the aspects of signals up to the boundary the first Interlocking needs to be regularly informed about the status of signals and track circuits in the fringe area of the other Interlocking. Distinct IDL telegrams are used to convey the intermittently and continuously needed data.

Outgoing IDL telegrams are prepared by commands in the Geographic Data and the generic control program is configured to copy their contents to the link at least once a major cycle. There are two parts to the remote route request protocol since it deals on the one hand with the problem of locking routes over Interlocking boundaries, and on the other, with releasing them again. The principles involved in achieving these

functions in SSI were introduced in Section 1.4, from where we recall the six main steps for the participants EAST and WEST in Figure 6.1. Both Interlockings use an elapsed timer to protect outgoing telegrams:

1. EAST receives a panel route request for a cross-boundary route. If the route is available in EAST, start the elapsed timer and issue a remote route request telegram to WEST.

2. WEST receives an IDL input conveying a remote route request. If the route is available, set the route, reply to EAST with an acknowledge telegram, and start the timer in order to guarantee transmission of the reply.

3. EAST receives a reply telegram to the earlier remote route request: if the timeout has not expired, EAST can then lock the route, stop the timer, and control the entrance signal as usual.

4. Whenever conditions indicate that a route has cleared up to the boundary, EAST issues a remote cancellation request to WEST as long as the elapsed timer is not running.

5. When WEST receives a request to cancel an inward route it does so unconditionally (but as long as its timer is not running), and acknowledges the request with a reply telegram to EAST.

6. On receipt of such an acknowledgement, EAST should cease to issue cancellation requests, the route having been cancelled in both Interlockings.

Because the precise conditions vary from route to route, the details of the protocol are implemented by standardised rules in the Geographic Data. Also, because of the desire to render invisible the subdivisions of the interlocking area under the control of one signal operator, the route release part of the protocol is implemented in the continuously executed *FOP* data, and not through panel requests. It is the signal engineer's responsibility to ensure that the elements of the protocol are correctly used, but who is to ensure that the protocol is "safe"?

In addressing this issue it is pertinent to ask what safety means in the context of this now distributed control system. At a human level of interpretation one could consider the control system safe if it is not prone to failure, there being an (essential) temporal component to the inquiry. However, since we deal here with a highly technical artefact—the control program, and the remote route locking algorithm it implements—the question can be sharpened. From [9]:

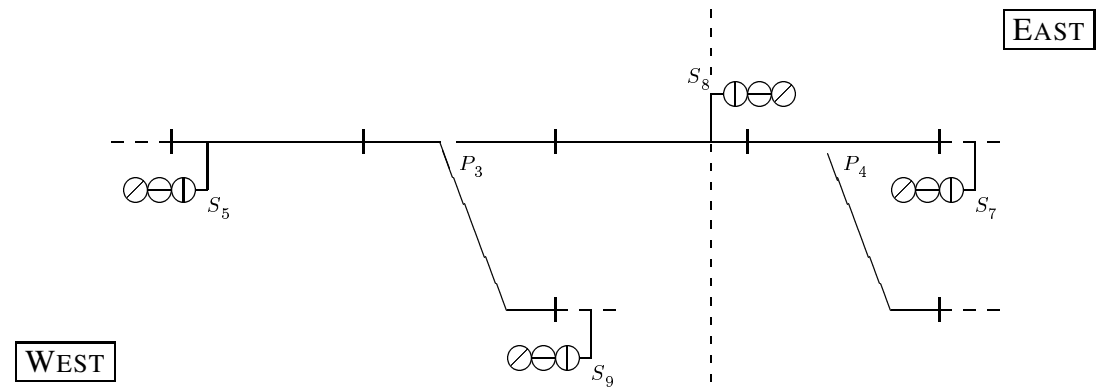


Figure 6.2: Derailment is likely if the tail part of R_{75} is not set

> *The overall requirement is to ensure that the [cross-boundary] route is set only if it is available in both Interlockings, and that it is never possible to arrive at a situation where half of the route only is set.*

For a precise discussion of safety the second clause should be refined. In practice the safety requirement is that it is never possible to arrive at a situation where only the first half of the route is set. In the sequel we shall insist that:

- *The remote route request protocol ensures the cross-boundary route is locked only if it is available in both Interlockings*, and
- *it is never possible to arrive at a situation where the first half only is locked.*

This more rigorous formulation of the safety requirement reflects two things: firstly, route *locking* is what the protocol is supposed to achieve, not route setting; secondly, it is assumed here as earlier that for a route to be *locked* is a necessary condition for the route to be *set*—that is, for the entrance signal to be switched off (in practice this means we should verify that the output to each signal drives the signal to red unless some onward route is locked). The conditions under which the entrance signal will be switched off depend on continually transmitted data from the other Interlocking, but those data are not transmitted as part of the remote route request protocol.

In order to appreciate the gravity of the situation in which only the first half of a cross-boundary is locked, consider the scheme in Figure 6.2. Suppose R_{75} is locked in EAST, but free in WEST, all signals are on, and tracks clear. If the route is set, S_7 can now go off to admit a train into the track section down to S_5. In the meantime, however, WEST can *lock* the route from S_9 which involves moving the points (P_3) normal—but the train approaching from EAST requires these points reverse and will (likely) be derailed if they are normal. Other hazards may also arise, but a head on collision is unlikely if the signals are properly interlocked according to the dynamic signalling rules (see Nock [76], and page 23).

It turns out, in fact, that under somewhat adverse conditions the protocol will fail to meet the second of the stated objectives. This is illustrated in Section 6.2.3, after a detailed description of how the Geographic Data implement remote route locking. In order to properly assess whether the technical violation of safety presents a real hazard, a more rigorous understanding of remote route locking is necessary. Therefore in Section 6.3 a formal model of the inter-SSI communications is developed that is based on the model in Chapter 3 which began our analysis. In fact, we shall simply extend Model #2 with the apparatus needed for two Interlockings to communicate—input and output buffers, telegrams, and timers.

The formal model serves a second, more important purpose, since it provides a framework within which it can be *proved* that the remote route request protocol can be safely implemented. In Section 6.4 we therefore develop the CCS model through a succession of refinements, the properties of each of which are verified using the model checker provided by the Concurrency Workbench. It turns out that safety *can* be assured as long as an additional timer is introduced to the protocol implemented by both parties in the negotiation. Unfortunately, however, this is not sufficient for safety—because of a second failure mode in the protocol. This problem is described, with its solution, in Section 6.4.3. The likely impact of our findings on the SSI are deferred until the concluding Section 7.1.

6.2 The Remote Route Request Protocol

In Section 1.4 the essential actions that are needed to lock cross-boundary routes were described, along with the elements that are required for their implementation. We briefly review how the SSI generic software acts so as to transport IDL telegrams, before spelling out the details of the Geographic Data that implement the protocol. Further details emerge as they become relevant in Section 6.3.

6.2.1 Preliminaries: Elapsed Timers and Telegrams

With the current generation of SSI the number of IDL telegrams that can be used is limited to a maximum of fifteen in total (so an SSI can exchange data with at most this many neighbours). The Interlockings connected to the link take it in turns to transmit all fifteen bytes of data (in a round-robin protocol), and the transport layer is configured so that each SSI can broadcast its data at least once a major cycle. These data are typically a mixture of messages containing signal and track circuit data, and request codes. The former are messages that are prepared from the *OPT* data file; the latter are prepared by commands from other data files (see Section 6.2.2 below). The *null* request code will often be transmitted (since usually there is no remote route request in

progress)—but null requests are ignored by the receiver. The SSI generic program is configured to copy non-null request codes to the IDL on two successive occasions: this provides fault tolerance in case the link is lossy, and ensures that like all other panel requests, IDL requests are processed twice.

An elapsed timer is associated with each IDL telegram that conveys request codes to another Interlocking (by convention). Timers may be *stopped* when not in use, or *running*—in which case they indicate an integral value in the range [0–254]. Timers are initialised by commands in the Geographic Data, usually by setting them to zero: thereafter they count upwards to a maximum value at which they stick (254), but may be *stopped* at any time by setting them to 255. The SSI generic program (not through the data) updates all such software timers in rotation, approximately once a second, but cannot update a timer more than once a major cycle. Software timers are only accurate to ± 2 s, so cannot be used for very precise timing. Given the strict polling cycle maintained by the SSI generic program it is evident that much more accurate system timers exist, but these are not meaningful to, or accessible from, the data.

Just as each *OPT* telegram is paired with an *IPT* telegram in the ongoing exchange of messages between SSI and track-side functional modules, so each *OPT* internal data link telegram is paired with an *IPT* telegram. When the telegram pair is used to convey request codes and their acknowledgements there will in fact be no *OPT* data (the code to transmit is calculated elsewhere, as we shall see), and the *IPT* data will simply specify the relationship between incoming request code and the desired panel request (in the *PRR* data file). On processing an incoming IDL telegram the interpreter will place the indicated panel request in the input buffer behind any outstanding panel requests.

The decisions taken at each step are programmed in the Geographic Data, and not in the SSI generic program which manages the low-level concerns of transferring requests to the internal data link, reading from the link, and queuing the appropriate panel requests for later processing. That is, the generic program provides the lower layers of the inter-SSI communications protocol on top of which the route request protocol is implemented. It is the responsibility of the signal engineer to prepare the Geographic Data in accordance with the guidelines described in [9]. The principles involved are now illustrated by considering the data necessary for the example in Figure 6.1.

6.2.2 Geographic Data

The Geographic Data needed to set up and cancel cross-boundary routes will be essentially similar to the example considered here, but the precise details will vary for every such route as the availability conditions will differ. In the sequel the identifiers G_W and G_E are used to represent the IDL telegram in EAST and WEST respectively

(the subscript identifies the recipient's address). The corresponding timers are E_W and E_E. To highlight where the telegram is written by a data command we shall use the form G_W =`Q75`, where `Q75` is the request code transmitted—pronounced 'send `Q75` to WEST'. Suppose EAST wishes to set the route from signal S_7:

1. When EAST receives a panel request for the route from the signal operator (via the control panel), all the route availability conditions specified in the *PRR* data are evaluated. Among these conditions is a test on the elapsed timer for the telegram used to convey request codes to WEST. Instead of locking the route, the command in the rule causes the timer to be started and the outgoing telegram to be written with the request code for the tail portion of the route:

   *Q75 if P_4 crf, T_9^{ca} f, E_W =stop
   then E_W =0, G_W =`Q75` \ .

   Here the timer is started by setting its value to zero. If no acknowledgement is received from WEST the timer will be stopped by a rule in the *FOP* data when it shows five (or more) seconds have elapsed:

   E_W =stop if E_W >5 \ .

2. When WEST receives a remote input from EAST, the code will be interpreted and the appropriate panel request queued. This is later evaluated to determine the availability of the tail portion of the route in WEST's interlocking area. Not only must the usual conditions prevail in the network, but the elapsed timer for the reply telegram should be stopped:

   *Q75 if T_7^{ba} f, E_E =stop
   then T_7^{ab} l, Z_7^{ab} l, E_E =3, G_E =`A75` \ .

   Z_7^{ab} is a dummy sub-route used in the sub-route release mechanism for cross-boundary routes. All EAST to WEST routes over this line will share this control variable. Here the elapsed timer is started (with an initial value of 3) in order to protect the acknowledgement to EAST, guaranteeing its transmission. The timer will be stopped by a rule appearing in the *FOP* data similar to that in EAST:

   E_E =stop if E_E >5 \ .

3. When EAST receives a remote input from WEST, the code will be interpreted and the appropriate panel request queued. When this is the acknowledgement to `Q75` the rule invoked reevaluates the availability of the route and checks that the request has not expired:

```
*A75 if P4 crf , T9ca f , EW <5
        then R75 s , P4 cr , T9ac l , T8ab l \
     Z8ab l , EW =stop .
```

The test 'E_W <5' fails if the timer has stopped. Whether or not the route can still be set the dummy outward sub-route (Z_8^{ab}) should be locked and the elapsed timer stopped.

4. Locking the dummy sub-route is necessary so the SSI will invoke the sub-route release request for the tail portion of the route (otherwise it remains locked in WEST). This rule resides in the *FOP* data and is therefore evaluated once every major cycle:

   ```
   if Z8ab l , T8ab f , EW =stop then GW =[QZ] \ .
   ```

   Whenever these conditions prevail the cancellation request will be issued—but note that this request is not protected by starting the timer. A subsequent request to WEST in the same major cycle may overwrite the cancellation request before it is posted.

5. Whenever WEST receives a request to cancel the inward portion of one of the cross-boundary routes it does so unconditionally, except for a timer test, by a panel request rule of the form:

   ```
   *QZ if EE =stop then Z7ab f , GE =[AZ] \ .
   ```

   Again note that the timer is not started to protect the reply telegram. The other sub-routes along the route in WEST will clear in the usual manner, but the first inward sub-route should always test the dummy inward sub-route:

   ```
   T7ab f if T7 c , Z7ab f \ .
   ```

6. Whenever EAST receives the acknowledgement to its request to cancel the tail portion of the outward route, the *PRR* rule frees the dummy outward sub-route only if the last sub-route up to the boundary is (still) free:

   ```
   *AZ if T8ab f then Z8ab f \ .
   ```

   There will normally be several cross-boundary routes, but there is only one dummy inward and one dummy outward sub-route needed in the protocol to cancel these routes (unless the boundary passes through points).

A number of observations follow from the foregoing description. Firstly, there is a simple priority mechanism that gives precedence to telegrams originating in the

PRR data over those arising in the *FOP* data. This is achieved by ensuring that only the *PRR* data may start the elapsed timer, while for all classes of data a test that it is stopped should be passed before writing an outgoing telegram. Secondly, the practice of not prioritising *FOP* data uses of the telegram means that the acknowledgement to a sub-route release request may be overwritten by a subsequent panel route request—or indeed by a request generated from the *FOP* data (if a route in the opposite direction is about to be released, say). This will leave the Interlocking initiating the cancellation request with the dummy outward sub-route locked even though the tail portion of the route is now cancelled. The first SSI will reissue the cancellation request until such time as it receives a reply from the second (or an outward route over the same line is again locked).

6.2.3 Safety Considerations

As a consequence of the lower priority given to *FOP* uses of the IDL telegram it is possible, where route locking in both directions across the boundary is required, for the inward and outward dummy sub-routes in one SSI to be locked simultaneously. No safety critical functions of the SSI should depend on the mutual exclusion property (**MX**) for these dummy variables; of course, the real sub-routes over the fringe track section should satisfy this condition. The routes property (**RT**), for both parts of the route, should also hold. In the case of the SSI controlling the tail portion of the route the property will state that *if the first inward sub-route is locked then the rest of the route is locked* (route variables for the tail parts of these cross-boundary routes are in fact defined, but they do not serve a route locking rôle, and they are not elements of the remote route request protocol). For the coupled system the required invariant states that if the first part of the route is locked then the tail portion is also locked.

Consider the normal sequence of events depicted graphically in Figure 6.3. The initiating panel request is processed in EAST (top rail), the timer started and shortly thereafter the telegram is placed on the IDL. WEST reads this message and places the request in its input buffer. When this is processed (successfully) the timer is started and the reply telegram written. EAST later reads the reply telegram from WEST and queues the appropriate second part of the panel request. When this is subsequently processed the route is locked in EAST, and the timer stopped. The timer in WEST expires because of the timeout implemented in the *FOP* data. If the remote route request fails in WEST the second timer is not started, and the first expires in due course.

The scenario sketched presupposes that IDL telegrams will be transmitted at most one major cycle after they were written (and the timer started). The ticks on the timelines indicate seconds—*i.e.*, moments at which the timers are updated by the real-time software. The major cycle can be no longer than the space between the ticks. In

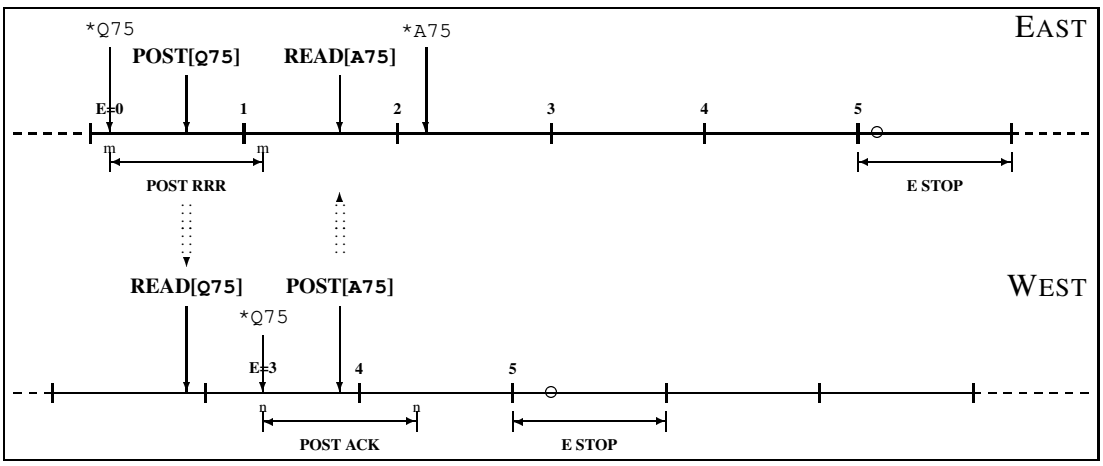


Figure 6.3: Normal sequence of events in making remote route requests

normal operation the whole process can easily be completed before the timer in EAST reaches the second tick.

Attention is drawn to the unusual sequence of events depicted in Figure 6.4. Matters are as before except for the insertion of some lengthy delays. The illustration suggests that the acknowledgement rule in the *PRR* data in EAST (*i.e.*, `*A75`) can be executed *after* the timeout has occurred. Of course, if this happens the timer test in the second rule fails, and so the route will not be locked in EAST—the usual (sub-route) release mechanism will free the tail portion locked in WEST. Indeed, the tail portion of the route may already have been released since the appropriate rule resides in the *FOP* data (the right circumstances for this can arise if, for instance, a route is being overset while a train is *en route*).

Unfortunately this is not the only action that can intervene between the timeout and processing the reply from WEST. In the lower part of Figure 6.4 this segment of EAST's timeline has been magnified, revealing a second panel request. This gives rise to concern only if the panel request restarts the elapsed timer: under these circumstances the delayed reply will then succeed, other things being equal, even though it should have expired. This leaves the route locked in EAST but free in WEST.

The question is whether these extremely long delays in processing panel route requests, delays that are of the order of several major cycles, are credible? Such delays are unlikely, but there are circumstances in which they could arise:

- If for some reason a data packet on the IDL becomes corrupted the receiving SSI will treat the telegram as if it were zero—the second transmission of the same request code will therefore afford a degree of fault tolerance. However, if the Interlockings exchange data only once a major cycle, and if the major cycle is long (closer to 1,000 ms than to 608 ms), delays will accumulate.

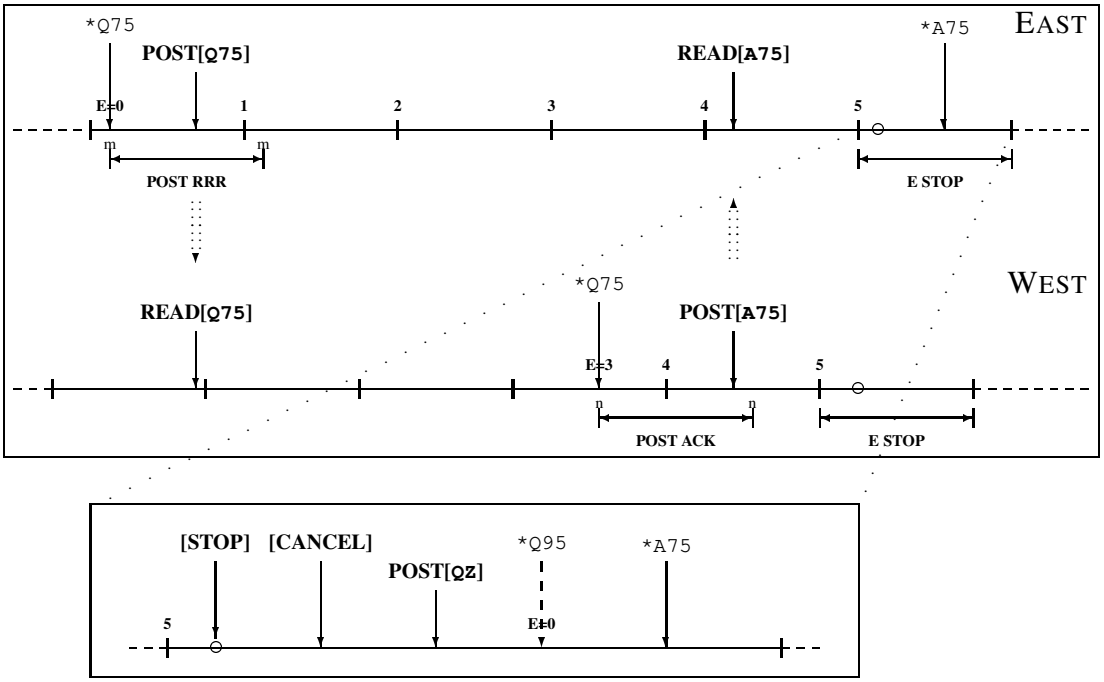


Figure 6.4: Abnormal sequence of events in making remote route requests

- If a route traverses complex point work in either Interlocking the route request may have to be split over several minor cycles. This can be achieved by dividing the route availability checks between several panel requests that follow one another: if the first succeeds, a second is queued; if the second succeeds a third is queued, and so on. This procedure will delay a panel request over several minor cycles.

- More seriously, a long route may straddle more than one Interlocking boundary, and more than one SSI may have to cooperate in locking a route, selecting and locking an overlap, and guaranteeing flank protection. If an intermediate SSI should need to make a further remote route request to set the tail, then even under favourable conditions several major cycles can elapse before the originating SSI receives the final acknowledgement.

Recall that queued panel requests will remain in the buffer for an indeterminate period of time because the interpreter will only process a panel request if all the required processes have been completed in under the minimum minor cycle time of 9.5 microseconds (see Section 1.3.2). Queued requests, whether from the control panel or received from another Interlocking, may therefore remain in the input buffer for arbitrary periods of time—but usually not more than a few minor cycles.

In order to assess whether the weakness in the protocol identified above presents a real hazard to railway traffic we shall need a more thorough analysis, and a formal model. On the one hand the purpose of such a model is to show precisely under what

circumstances this fault will be manifest (for example, by making explicit all our assumptions about the SSI's behaviour). On the other hand it is to place our understanding on a more rigorous foundation so as to formulate strategies to overcome the problem and prove, within the framework of the model, that safety can be assured.

6.3 Modelling Remote Route Locking

Evidently a model of these aspects of the inter-SSI communications needs some notion of time. The CCS model of Chapter 3 was devised to examine static properties of the Geographic Data where timing issues were not considered—thus Model #2 admits no notion of a major cycle, nor even a minor cycle, by which to clock the system. In developing that model further here, we therefore introduce such a clocking mechanism.

6.3.1 Timing Issues

Precise arguments about critical timing properties of systems cannot be made within the framework of an asynchronous calculus such as CCS whose semantics say nothing about the duration of actions, nor the duration of the intervals between them. Such properties can be explored using languages such as Timed CCS [71] and Timed CSP [82] which introduce discrete or continuous domains of 'time transitions'. Generally, either time progresses or computation does, but not both together.

Where the events that clock a system are discrete, for example in synchronous hardware where a global clock signal defines the frequency with which all system components change state, the weaker time model of SCCS [59] or MEIJE [27] may offer the right kind of temporal abstraction. In such models all system components proceed in lockstep whether or not they communicate, computation always coinciding with the clock's tick. From the synchronous model of process interaction one may, as Milner showed, recover the asynchronous behaviour associated with distributed systems.

The natural clock of the SSI is the *major* cycle. Not only does this define the frequency with which the Interlocking and the track-side modules exchange fresh data, but it also defines the maximum frequency at which the system's elapsed timers can be updated. This latter point is crucial since the behaviour associated with the remote route request protocol does not depend on the number of seconds that have passed since the outgoing IDL telegram was written, but on the number of times the elapsed timer has been updated. We need not, therefore, introduce real time to the model *a priori*.

6.3.2 A Formal CCS Model

The Interlockings connected to the internal data link are not tightly synchronised with one another. The lower layers of the inter-SSI communications protocol ensure that

each SSI communicates its outgoing telegram data to the bus at least once a major cycle. For the formal model we shall tighten this assumption and suppose that each SSI communicates exactly once (twice, *etc.*,) a major cycle. These communication events are used to clock the system. We shall further assume, grossly erring on the side of pessimism, that one major cycle consumes one second of elapsed time. Since 608 ms $\leq$ 1 major cycle $\leq$ 1 elapsed 'second', the second inequality will be taken to be a strict equality, and the model's elapsed timers will therefore count major cycles.

To simplify the presentation of the model let us suppose that only two Interlockings are connected to the link (*i.e.*, the effects of the other Interlockings connected to the link will be ignored). Extending Model #2, the elements needed include an elapsed timer, an input queue, and an output buffer to hold the current output telegram. We do not model the internal data link *per se*, merely wishing to count the synchronisations between the two Interlockings. Any lossy behaviour of the link can be emulated in the input or output buffers.

Elapsed Timers In SSI, elapsed times that protect IDL telegrams used to convey request codes should always be stopped by a timeout command in the *FOP* data (in case they are not stopped by rules in the *PRR* data). This timeout behaviour is modelled here with a watchdog timer which may be started or reset by the data, but which is otherwise stopped by an external event. This simple version increments the counter on each sync action (provided by the Control, below):

$$\mathsf{E}(m) \stackrel{\text{def}}{=} \begin{cases} \overline{\mathsf{get}}_E(m).\mathsf{E}(m) + \mathsf{put}_E(n).\mathsf{E}(n) + \mathsf{stop}.\mathsf{E}(\mathtt{stop}) & m = M \\ \overline{\mathsf{get}}_E(m).\mathsf{E}(m) + \mathsf{put}_E(n).\mathsf{E}(n) + \mathsf{sync}.\mathsf{E}(m+1) & 0 \leq m < M \\ \overline{\mathsf{get}}_E(m).\mathsf{E}(m) + \mathsf{put}_E(n).\mathsf{E}(n) + \mathsf{sync}.\mathsf{E}(m) & m = \mathtt{stop} \end{cases}$$

The timer cannot synchronise again once it has counted up to M: it must stop explicitly by the stop action, or be reset from the data via $\mathsf{put}_E(\mathtt{stop})$. A stopped timer will continue to synchronise as required. Let $M < \mathtt{stop}$ be the maximum timeout the timer can observe (*e.g.*, five 'seconds'). This formulation reflects the notion that while the timeout is programmed in the *FOP* data, it is undesirable to specify in which minor cycle the relevant rule is to be executed—because, although the execution order is fixed, it is arbitrary and so we should therefore assume nothing about it. The timeout is thus represented by the stop action which *must* occur in the $(M-n)^{\text{th}}$ cycle after the timer was started with the value n. The above definition is simple to generalise so that the timer can be incremented instead on every second, third, *etc.*, sync action.

Queue Process The specification of the input ring-buffer is data orientated. Let $[\,]$ represent the empty queue, and $[e \mid q]$ represent the queue whose first element is e and

whose tail is q. This agent will synchronise with Control in recording IDL inputs, through qin, and again in removing panel requests, through set. Other input requests, *e.g.*, from the control panel, will enter the queue directly (via qin).

$$\mathsf{Q}(q) \stackrel{\text{def}}{=} \begin{cases} \mathsf{qin}(e).\mathsf{Q}(app_l(e,q)) & \text{if } q = [\,] \\ \mathsf{qin}(e).\mathsf{Q}(app_l(e,q)) + \overline{\mathsf{set}}(e').\mathsf{Q}(q') & \text{if } q = [e' \mid q'] \end{cases}$$

Here e, $e' \in \mathcal{Q}$, where $\mathcal{Q}$ is the set of all panel requests, there being one rule in the *PRR* data for each element of $\mathcal{Q}$. In the sequel we let 0 represent the null telegram, and for convenience let $0 \in \mathcal{Q}$.

The append function, app_l, places the new arrival at the end of the queue as expected. The specification should be made more precise since the buffer will be of bounded capacity, say l:

$$app_l(e,q) = \begin{cases} q & \text{if } e = 0 \text{ or } length(q) = l \\ app(e,q) & \text{otherwise} \end{cases}$$

$$app(e,q) = \begin{cases} [e \mid q] & \text{if } q = [\,] \\ [e' \mid app(e,q')] & \text{if } q = [e' \mid q'] \end{cases}$$

An input arriving at a full queue will be silently discarded.

Telegram In the Geographic Data, outgoing telegrams are treated just as any other variable. SSI normally resets a telegram's contents to zero when it is posted. Here, we model a telegram as a process which can hold one of a set $\mathcal{O}$ of values (also letting $0 \in \mathcal{O}$) which can occasionally be placed on the link:

$$\mathsf{G}(u) \stackrel{\text{def}}{=} \overline{\mathsf{get}}_G(u).\mathsf{G}(0) + \mathsf{put}_G(v).\mathsf{G}(v) \quad u \in \mathcal{O}$$

The telegram is reset to zero whenever its contents are read. The (only) process to read this variable is the output buffer which is a simple cycler (for the moment, it is refined later) that executes the sequence: output to the IDL, await the next enabling action (turn) in accordance with the round-robin protocol, and fetch the next output telegram:

$$\mathsf{O}(u) \stackrel{\text{def}}{=} \overline{\mathsf{idl_out}}(u).\mathsf{turn}.\mathsf{get}_G(v).\mathsf{O}(v) \quad u \in \mathcal{O}$$

In general, if there are $n+1$ Interlockings connected to the IDL the output buffer should await n enabling actions before being permitted to communicate the next output. Here, any lossy behaviour associated with the link, and the compensating duplication of telegrams, has been ignored.

Control The Control is constructed just as in Chapter 3, but with an explicit synchronisation (with the elapsed timer) to mark the start of the major cycle loop:

$$\mathsf{Control} \stackrel{\text{def}}{=} \mathsf{set}(\mathrm{Q}).(\mathbf{C}[\![\mathit{PRR}(\star\mathrm{Q})]\!]\mathsf{Control}) + \cdots + \mathsf{idl_in}(v).\overline{\mathsf{qin}}(v).\mathsf{Sync}$$

$$\mathsf{Sync} \stackrel{\text{def}}{=} \overline{\mathsf{sync}}.\overline{\mathsf{turn}}.\mathsf{Control}$$

Now, in addition, the Control reads an IDL input, queues the appropriate panel request and updates the major cycle counter. Having read an IDL input, the Control also enables the output buffer for writing to the link. The translation $\mathbf{C}[\![\cdot]\!]$ is extended canonically. The Control can now be composed with the output buffer and the timer, *e.g.*, as in

$$((\mathsf{Control} \mid \mathsf{O}(u))\backslash\{\mathsf{turn}\} \mid \mathsf{E}(m))\backslash\{\mathsf{sync}\}$$

so that the model is 'clocked on input'—that is, we increment the timer on each input from the other Interlocking, then enable the output telegram. Since both Interlockings implement the same protocol, this ensures a (logical) one cycle delay between outputs, or inputs, and gives the model a simple clocking mechanism. Note that the Control reads the IDL and queues the input for later processing. Other inputs (*i.e.*, those received from the control panel) enter the queue directly.

6.3.3 Matching up the Interfaces between East & West

The model is then specified by the parallel composition of agents

$$(\mathsf{Control} \mid \mathsf{Image} \mid \mathsf{Q}([\,]) \mid \mathsf{E}(0) \mid \mathsf{G}(0) \mid \mathsf{O}(0))\backslash L$$

whose components are gathered together in Model #3. Each component captures a separate function of the generic program as discussed above. The restriction set, L, is chosen so that the visible actions include

$\{\mathsf{idl_in}(m) \mid m \in \mathcal{I}\}$	inputs from the other Interlocking(s),
$\{\overline{\mathsf{idl_out}}(m) \mid m \in \mathcal{O}\}$	outputs to the other Interlocking(s),
$\{\mathsf{qin}(m) \mid m \in \mathcal{Q} - \mathcal{I}\}$	panel requests from the signal control panel,

along with $\{\mathsf{obs}_D(v) \mid D \in \mathcal{D}\}$, the observers (tags) in Image, stop, the timeout for the elapsed timer, and the other visible inputs to Control discussed in Section 3.2. $\mathcal{I} \subseteq \mathcal{Q}$ (with $0 \in \mathcal{I}$) is the set of IDL inputs the SSI can receive. We then compose two such systems, say East and West, in such a way as to ensure that they synchronise on the IDL transmissions:

$$\begin{array}{rcl} \mathsf{EastWest} & \stackrel{\text{def}}{=} & (\mathsf{East}[\mathsf{idl_in}(m)/\mathsf{idl_out}(m) \mid m \in \mathcal{O}_E][\mathsf{stop}_E/\mathsf{stop}] \\ & \mid & \mathsf{West}[\mathsf{idl_in}(m)/\mathsf{idl_out}(m) \mid m \in \mathcal{O}_W][\mathsf{stop}_W/\mathsf{stop}]) \\ & \backslash & \{\mathsf{idl_in}(m) \mid m \in \mathcal{O}_W \cup \mathcal{O}_E\} \end{array}$$

The set $\mathcal{O}_W$ ('out WEST', which is the same as $\mathcal{I}_E$) is the union of two disjoint sets of messages: the requests sent from West to East, and the set of West's replies to

$$
\begin{array}{rcl}
\mathsf{E}(m) & \stackrel{\text{def}}{=} & \begin{cases} \overline{\mathsf{get}}_E(m).\mathsf{E}(m) + \mathsf{put}_E(n).\mathsf{E}(n) + \mathsf{stop}.\mathsf{E}(\mathtt{stop}) & \text{if } m = M \\ \overline{\mathsf{get}}_E(m).\mathsf{E}(m) + \mathsf{put}_E(n).\mathsf{E}(n) + \mathsf{sync}.\mathsf{E}(m+1) & \text{if } 0 \le m < M \\ \overline{\mathsf{get}}_E(m).\mathsf{E}(m) + \mathsf{put}_E(n).\mathsf{E}(n) + \mathsf{sync}.\mathsf{E}(m) & \text{if } m = \mathtt{stop} \end{cases} \\
\mathsf{Q}(q) & \stackrel{\text{def}}{=} & \begin{cases} \mathsf{qin}(e).\mathsf{Q}(app_l(e,q)) & \text{if } q = [\,] \\ \mathsf{qin}(e).\mathsf{Q}(app_l(e,q)) + \overline{\mathsf{set}}(e').\mathsf{Q}(q') & \text{if } q = [e' \mid q'] \end{cases} \\
\mathsf{G}(u) & \stackrel{\text{def}}{=} & \overline{\mathsf{get}}_G(u).\mathsf{G}(0) + \mathsf{put}_G(v).\mathsf{G}(v) \\
\mathsf{O}(u) & \stackrel{\text{def}}{=} & \overline{\mathsf{idl_out}}(u).\mathsf{turn}.\mathsf{get}_G(v).\mathsf{O}(v) \\
\mathsf{Control} & \stackrel{\text{def}}{=} & \mathsf{set}(\mathrm{Q}).(\mathbf{C}[\![\mathit{PRR}(\star\mathrm{Q})]\!]\mathsf{Control}) + \cdots + \mathsf{idl_in}(v).\overline{\mathsf{qin}}(v).\mathsf{Sync} \\
\mathsf{Sync} & \stackrel{\text{def}}{=} & \overline{\mathsf{sync}}.\overline{\mathsf{turn}}.\mathsf{Control} \\
\mathsf{SSI} & \stackrel{\text{def}}{=} & (\mathsf{Control} \mid \mathsf{Image} \mid \mathsf{Q}([\,]) \mid \mathsf{E}(0) \mid \mathsf{G}(0) \mid \mathsf{O}(0)) \backslash L
\end{array}
$$

$$
\begin{array}{rcl}
app_l(e,q) & = & \begin{cases} q & \text{if } e = 0 \text{ or } length(q) = l \\ app(e,q) & \text{otherwise} \end{cases} \\
app(e,q) & = & \begin{cases} [e \mid q] & \text{if } q = [\,] \\ [e' \mid app(e,q')] & \text{if } q = [e' \mid q'] \end{cases}
\end{array}
$$

Model #3: Simple model of SSI communications over Internal Data Link

the requests received from East. $\mathcal{O}_E$ is similarly defined. We arrange that the sets of observations (tags in Image) in the two systems are disjoint. In particular

$\{\mathsf{qin}(m) \mid m \in (\mathcal{Q}_E - \mathcal{I}_E) \uplus (\mathcal{Q}_W - \mathcal{I}_W)\}$	panel inputs for East or West,
$\{\mathsf{eobs}_D(v) \mid D \in \mathcal{D}_E\}$	observations (tags) in East,
$\{\mathsf{wobs}_D(v) \mid D \in \mathcal{D}_W\}$	observations (tags) in West,

are visible actions of the coupled system, along with $\{\mathsf{stop}_E, \mathsf{stop}_W\}$, the timeout actions for the elapsed timers. The sets $\mathcal{D}_E$ and $\mathcal{D}_W$ are the control variables (points, routes, sub-routes and track circuits) defined in each Interlocking.

Having set up the model as above it should be remarked, before proceeding with the analysis, that the semantics of CCS do not at all compel the exchange of data between East and West to take place. Indeed, both systems can proceed independently without ever synchronising—but then in neither frame of reference will time advance.

6.3.4 Axiomatising Remote Route Requests

Although describing only the binary case, the formal CCS model described above is couched in rather general terms. In fact, the protocol itself will have to be instantiated (in the Geographic Data) for each Interlocking boundary—that is, in preparing the Data for each SSI, a telegram (G) and an elapsed timer (E) have to be allocated

1. `*QN if @CN,` E `=stop then` E `=0,` G `=`$\boxed{\texttt{QN}}$ `\ .`
2. `*AN if @CN,` E `<5 then @SN,` T_L^{out} `l \` Z_L^{out} `l,` E `=stop .`
3. `*QM if @CM,` E `=stop then @SM,` T_L^{in} `l,` Z_L^{in} `l,` E `=3,` G `=`$\boxed{\texttt{AM}}$ `\ .`
4. `*AZ if` T_L^{out} `f then` Z_L^{out} `f \ .`
5. `*QZ if` E `=stop then` Z_L^{in} `f,` G `=`$\boxed{\texttt{AZ}}$ `\ .`
6. E `=stop if` E `>5 \ .`
7. `if` Z_L^{out} `l,` T_L^{out} `f,` E `=stop then` G `=`$\boxed{\texttt{QZ}}$ `\ .`
8. T_L^{in} `f if` Z_L^{in} `f,` T_L `c \ .`

Figure 6.5: Generic rules for remote route locking and release

for each boundary over which routes must be set, and the data for the individual routes should adhere to the guidelines sketched in Section 6.2. If L is just one of the lines over the SSI boundary where routes must be set, there are potentially eight generic rules in the *PRR* and *FOP* data that will be instantiated (see Figure 6.5). Of these, rules (1) and (2) will be instantiated for each outward route N; rule (3) will be instantiated for each inward route M; the other rules need instantiating only once for line L, except rule (6) which needs instantiating only once for each timer/telegram. No other Geographic Data should update the telegram, the timer, or the dummy sub-routes needed to implement remote route locking. The availability conditions for a route, and the commands for locking a route, are inserted at the place-holders `@CN`, `@SN`, *etc.*. Note that the *real* inward and outward sub-routes are mentioned in rules (2) and (3). Here we assume that the boundary is on plain track, and that no route straddles more than one SSI boundary.

Ideally one would like to establish the safety of the remote route request protocol independently of the safety analysis required for the rest of the Geographic Data. Unfortunately this is not possible since errors in the route specific data (`@CN`, and `@SN`), may introduce unsafe states irrespective of the correct functioning of the protocol. So we assume the data are correct—this is valid because the safety properties of the Geographic Data can be established independently of the 'correctness' of the protocol. For example, the safety property **F** of the earlier chapters makes no reference to the additional control data needed to implement remote route locking, so theorems like $\{\mathbf{F}\}\, c_i\, \{\mathbf{F}\}$, for $i = 1 \ldots 8$ above, are rather easy to prove.

In analysing properties of the protocol it is therefore reasonable to assume that the specific route data satisfy the appropriate invariants—**RT** in particular, with the characterisation given in Section 5.3.3: if the route is locked, all the sub-routes on the route are locked; if the first sub-route on the route is locked, the remaining sub-routes are locked, and so on. This considerably simplifies of the model, leaving only the generic parts listed above, and omitting the specific route locking data entirely.

Intuitively, the route locking conditions will always be passed, and the only action taken in locking a cross-boundary route (other than executing the protocol actions of course) is to lock the inward or outward sub-route as appropriate. Where routes are set in the direction EAST to WEST only, we require rules (1), (2), (4), and (7) in East, and (3), (5), and (8) in West. The timeout rule (6) is omitted since the watchdog mechanism is used instead.

This, then, is the formal model submitted to the Concurrency Workbench for semantic analysis. The interface between the two components is specified by the sets $\mathcal{I}_E = \{\mathtt{AN}, \mathtt{AZ}\}$ and $\mathcal{I}_W = \{\mathtt{QN}, \mathtt{QZ}\}$ while $\mathcal{Q}_W = \mathcal{I}_W$ and $\mathcal{Q}_E = \mathcal{I}_E \uplus \{\mathtt{QN}, \mathtt{QNX}\}$. Apart from the timeout actions associated with the elapsed timers, stop_E and stop_W, the only inputs of interest are in East: $\mathsf{qin}(\mathtt{QN})$, and this route's (unconditional) cancellation $\mathsf{qin}(\mathtt{QNX})$. Outputs are just the observations made of the control variables T_L^{out}, Z_L^{out}, T_L^{in} and Z_L^{in}.

6.4 Safety Properties of the Model

Informally, the safety property associated with the remote route request protocol can be expressed thus:

> *whenever the initial portion of a cross-boundary route is locked in the first Interlocking, the tail portion of the route is locked in the second.*

In light of the discussion in Section 5.3.3 we strengthen this to:

> *whenever the last sub-route of the first portion of the route is locked, the first sub-route of the tail portion is locked.*

If this property holds it also ensures that the route is not prematurely released in the second Interlocking. The modal formula Θ, where

$$\Theta \stackrel{\text{def}}{=} \langle \mathsf{eobs}_{T_L^{out}}(\mathtt{l}) \rangle tt \wedge \langle \mathsf{wobs}_{T_L^{in}}(\mathtt{f}) \rangle tt$$

therefore characterises the unsafe states in EastWest. This is sufficient since, by hypothesis, each Interlocking independently satisfies **RT**, and in particular WEST has the property that if the first sub-route on the cross-boundary route is locked, then the rest of the (tail of the) route is locked. Now $\neg\Theta \Leftrightarrow \langle \mathsf{eobs}_{T_L^{out}}(\mathtt{l}) \rangle tt \Rightarrow \langle \mathsf{wobs}_{T_L^{in}}(\mathtt{l}) \rangle tt$, so the invariant all states of the model should satisfy is just $\Xi \stackrel{\text{def}}{=} \nu Z.\neg\Theta \wedge [-]Z$. When started from a sensible initial state, no reachable state of the model should satisfy Θ.

Given an initial state in which all the sub-routes are *free* the local model checker confirms that:

$\mathsf{EastWest}_{\#3}$	$\models$	$\nu Z.\langle - \rangle tt \wedge [-]Z$	: Freedom from deadlock,
$\mathsf{EastWest}_{\#3}$	$\models$	$\langle\!\langle \mathsf{qin}(\mathtt{QN}) \rangle\!\rangle \langle\!\langle \mathsf{eobs}_{T_L^{out}}(\mathtt{l}) \rangle\!\rangle tt$	: Can set the outward route,
$\mathsf{EastWest}_{\#3}$	$\not\models$	Ξ	: Safety.

$$
\begin{array}{rcl}
\mathsf{Q}(q) & \stackrel{\text{def}}{=} & \begin{cases} \mathsf{qin}(e).\mathsf{Q}(app_l(e,q)) & \text{if } q = [\,] \\ \mathsf{qin}(e).\mathsf{A}(e,q) + \overline{\mathsf{set}}(e').\mathsf{Q}(q') & \text{if } q = [e'_n \mid q'] \end{cases} \\
\mathsf{A}(e,q) & \stackrel{\text{def}}{=} & \textit{if } (e \in \mathcal{I}) \textit{ then } \mathsf{Q}(app_l(e, age(q))) \textit{ else } \mathsf{Q}(app_l(e,q))
\end{array}
$$

$$
\begin{array}{rcl}
app(e,q) & = & \begin{cases} [idl_\Delta(e) \mid q] & \text{if } q = [\,] \\ [e'_n \mid app(e,q')] & \text{if } q = [e'_n \mid q'] \end{cases} \\
age(q) & = & \begin{cases} q & \text{if } q = [\,] \\ age(q') & \text{if } q = [e_n \mid q'] \text{ and } n = 1 \\ [e_{n \dot{-} 1} \mid age(q')] & \text{if } q = [e_n \mid q'] \text{ and } n \neq 1 \end{cases} \\
idl_\Delta(e) & = & \begin{cases} e_\Delta & \text{if } e \in \mathcal{I} \\ e_0 & \text{otherwise} \end{cases}
\end{array}
$$

Model #4: Refining Model #3 so as to discard tardy IDL inputs after Δ cycles

The double diamond modality, $\langle\!\langle a \rangle\!\rangle$, abstracts from the silent or unobservable activity of the model (accompanying the initial input, in this case). If $\mathsf{EastWest}'_{\#3}$ is a state in which Z_L^{out} is *locked* but T_L^{out} is *free* (both timers being stopped), then

$$\mathsf{EastWest}'_{\#3} \quad \models \quad \langle\!\langle \mathsf{qin}(\mathtt{QN}) \rangle\!\rangle \langle\!\langle \mathsf{qin}(\mathtt{QN}) \rangle\!\rangle \langle\!\langle \mathsf{stop}_W \rangle\!\rangle \langle\!\langle \mathsf{stop}_E \rangle\!\rangle \Theta$$

The reason $\mathsf{EastWest}_{\#3} \not\models \Xi$ is just as discussed earlier in Section 6.2.3: the possibility that IDL inputs can remain in the queue indefinitely.

6.4.1 First Refinement: Eliminating Arbitrary Delays

If the problem arises from the possibility that panel requests can remain in the queue for an indeterminate period of time, perhaps the situation can be repaired by eliminating such arbitrary delays? This introduces the idea that IDL inputs should expire if they have been queuing "too long" (a parameter one might wish to adjust). This is achieved in Model #4 by modifying the queue process so that panel requests are given timestamps. In the data part of the specification of the queue, app adds input e with its timestamp to the queue—the function is defined much as before, but now elements will be (time) indexed. No attempt has been made here to make these specifications efficient, only precise.

Given that $\mathcal{Q}$ is the set of all panel requests the SSI can receive, and $\mathcal{I} \subseteq \mathcal{Q}$ are just the IDL requests, the function idl_Δ computes the initial timestamp for the given input e: this is non-zero if and only if $e \in \mathcal{I}$. The delay parameter Δ is chosen to be the maximum length of time (the number of major cycles) a message can have been pending before expiring. The function age is arranged so that tardy IDL requests are

discarded. A zero timestamp ensures that the queued request is not discarded however long it has been waiting. A non-zero timestamp should be interpreted as the number of cycles a request can remain in the queue.

A small change in the queue process is necessary since we only propose IDL inputs should expire. The clause

$$\mathsf{A}(e,q) \stackrel{\text{def}}{=} \textit{if } (e \in \mathcal{I}) \textit{ then } \mathsf{Q}(app_l(e, age(q))) \textit{ else } \mathsf{Q}(app_l(e,q))$$

ensures that the timestamps are only decremented when an IDL input is received. In generalising this scheme, Δ should be some constant multiple of the total number of IDL inputs the SSI receives (*i.e.*, those addressed to it) in one major cycle.

Setting $\Delta = 1$ the result from the model checker is that in addition to satisfying the deadlock freedom and liveness properties cited earlier, the model now satisfies the safety property: $\mathsf{EastWest}_{\#4} \models \Xi$. One can investigate further and set $\Delta = 2$, which is a more generous delay, but the result is somewhat surprising: since elapsed timers count up to five 'seconds' a combined delay (*i.e.*, in EAST and WEST) of four 'seconds' is long enough for states satisfying Θ to reappear. To understand this fully it is best to explore the behaviour of Model #4 more thoroughly—for which purposes we need a simulation environment to animate the formal model. The Concurrency Workbench can also be used for this purpose, but however the model is simulated one can derive a transition sequence such as that depicted in Figure 6.6.

At line (1) the initiating panel request is made. This is processed at line (2) and there follows a series of (ten) message exchanges. The remote route request must be processed in WEST before line (9), otherwise it expires. In the figure, 0 represents the null telegram, • the stopped state of the elapsed timer, and blank spaces in columns Q, E, and G indicate 'no change' (with respect to the line above). At line (17) we see the acknowledgement waiting to be processed in EAST, the timer is running, but the route is unset in WEST. The main point to observe here is that if the fault is to occur the second panel request, which need not of course be for the same route, nor even refer to the same line, must be queued before the reply telegram is received by EAST.

Moreover, also from the trace in Figure 6.6, we can infer that when $\Delta = 1$ the reply telegram will always arrive at EAST (if it arrives at all) before the elapsed timer has advanced to four. If this were not the case the elapsed timer could stop before the reply was processed, leading to a state satisfying Θ—but this cannot be the case since $\neg\Theta$ is invariant. This suggests that as long as the SSI can guarantee to process all panel requests within a single major cycle, the remote route request protocol is safe. This conclusion is in fact premature, as we shall see below when modelling IDL faults.

	EAST	Q	E	G		Q	E	G	WEST	
(1)	$\{\text{PRR}\}^{\rightarrow}$	$[\text{QN}_0]$	•	0		[ ]	•	0		(1)
(2)		[ ]	0	QN	$\longleftarrow$					(2)
(3)			1		$\longrightarrow$					(3)
(4)				0	$\longleftarrow$	$[\text{QN}_2]$				(4)
(5)			2		$\longrightarrow$					(5)
(6)					$\longleftarrow$	$[\text{QN}_1]$				(6)
(7)			3							(7)
(8)					$\longrightarrow$	[ ]	3	AN		(8)
(9)							4			(9)
(10)	$\{\text{PRR}\}^{\rightarrow}$	$[\text{QN}_0]$			$\longleftarrow$					(10)
(11)		$[\text{QN}_0 \mid \text{AN}_2]$	4		$\longrightarrow$			0		(11)
(12)					$\longleftarrow$		5			(12)
(13)		$[\text{QN}_0 \mid \text{AN}_1]$	5				•		$^{\leftarrow}\{\text{STOP}\}$	(13)
(14)	$\{\text{STOP}\}^{\rightarrow}$		•							(14)
(15)	$\{\text{FOP}\}^{\rightarrow}$			QZ	$\longrightarrow$					(15)
(16)				0		$[\text{QZ}_2]$				(16)
(17)		$[\text{AN}_1]$	0	QN		[ ]		AZ		(17)

Figure 6.6: Illustrating how unsafe states arise in Model #4 when $\Delta = 2$

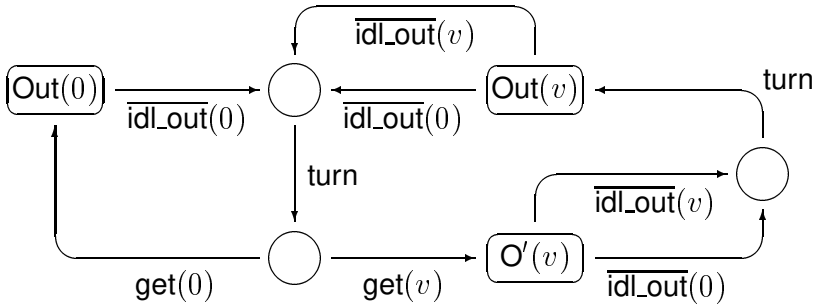
6.4.2 Second Refinement: Adding Priorities

In refining the model above we might observe that there seems to be no need to assign any special consideration to cancellation requests, nor to the acknowledgements to remote route or cancellation requests. That is, acknowledgements need not expire if the system is busy. Indeed, since the proposed solution to this flaw in the protocol sets the delay $\Delta = 1$, a busy Interlocking may sometimes find it difficult to lock cross-boundary routes. Let us therefore

- arrange to timeout only remote route *requests* and not their acknowledgements, nor cancellation requests, nor their acknowledgements, and
- assign higher priority to the messages acknowledging remote route requests in the queue process—*i.e.*, implement a priority queue.

It turns out that these modifications have to be implemented together for otherwise the model remains unsafe with respect to Θ even when $\Delta = 1$.

Given $\mathcal{I}$, two disjoint subsets can be identified: $\mathcal{Q}_I$ ('queries in'), the remote route requests served by this SSI, and $\mathcal{A}_O$, the acknowledgements to the remote route requests sent by this SSI. All other IDL inputs (*e.g.*, cancellation requests) are contained in $\mathcal{I} - (\mathcal{Q}_I \uplus \mathcal{A}_O)$. The queue process is defined as in Model #4, but in the data part of



Model #5: Introducing lossy link behaviour to Model #4

the specification the definitions of app and idl_Δ are replaced by those displayed below:

$$app(e,q) = \begin{cases} [idl_\Delta(e) \mid q] & \text{if } q = [\,] \text{ or } e \in \mathcal{A}_O \\ [e'_n \mid app(e,q')] & \text{if } q = [e'_n \mid q'] \text{ and } e \notin \mathcal{A}_O \end{cases}$$

$$idl_\Delta(e) = \begin{cases} e_\Delta & \text{if } e \in \mathcal{Q}_I \\ e_0 & \text{otherwise} \end{cases}$$

When setting $\Delta = 1$, 2, or 3 cycles $\mathsf{EastWest}_{\#4'} \models \Xi$, but higher values lead to unsafe states as before. Note that the priority mechanism ensures the acknowledgement is always processed before any other panel requests which could restart the elapsed timer (*i.e.*, the order of the elements in the queue in EAST at line (11) in Figure 6.6

6.4.3 Lossy Communications and Duplicating Telegrams

Model #3 and its refinements above are built upon the assumption that the communication medium, the internal data link, functions perfectly at all times. It has been shown therefore that even when this is the case a reasonable (though slightly pessimistic) model of the inter-SSI communications fails to satisfy the overall safety requirement. By eliminating arbitrary delays in the input buffer we have seen that safety can be assured (in the model). However, this is not the only, nor even the likely, source of delays which the SSI has to tolerate. Recall that IDL telegrams may occasionally be lost due to imperfections in the communications medium, and so the telegrams used to convey requests to other Interlockings are duplicated (once) to tolerate such faults.

If the two Interlockings exchange messages only once a major cycle the loss (or failure) of a remote route request, followed by the loss of the first reply, can lead to several seconds' delay. This weakness will be exacerbated if a third SSI has to enter the negotiations to set a route that straddles more than one Interlocking boundary. In order to model the lossy behaviour of the IDL the modifications shown in Model #5 are implemented (the transition diagram representation of the agent $\mathsf{O}(v)$ is preferred here as it is easier to interpret). In this specification the buffer process $\mathsf{O}(v)$ now duplicates

the value read from the telegram, through $\mathsf{O}'(v)$, but may nondeterministically transmit the null telegram instead of v. This models possible faults on the IDL. Note that in the left-hand loop null telegrams are not duplicated. In the model, neither Interlocking engaged in these communications will detect the faults occurring on the link.

Implementing these changes in Model #4 we can construct $\mathsf{EastWest}_{\#5}$, and setting $\Delta = 1$ once more interrogate the model checker:

$$\begin{array}{lcll} \mathsf{EastWest}_{\#5} & \models & \nu Z.\langle - \rangle tt \wedge [-]Z & : \text{Freedom from deadlock} \\ \mathsf{EastWest}_{\#5} & \models & \langle\!\langle \mathsf{qin}(\mathtt{QN}) \rangle\!\rangle \langle\!\langle \mathsf{eobs}_{T_L^{out}}(\mathtt{1}) \rangle\!\rangle tt & : \text{Can set the outward route} \\ \mathsf{EastWest}_{\#5} & \not\models & \Xi & : \text{Safety.} \end{array}$$

This result is no surprise when Figure 6.6 shows that a two cycle delay in the input buffer leads to unsafe states: a three cycle delay is also "too long".

That said, there is one parameter in the remote route request protocol that can be adjusted with ease: the hitherto arbitrary timeout observed by the elapsed timers, currently set to five major cycles. It is now possible to suggest much more precisely what value this should take if the protocol is to ensure both safe and live usage of the IDL. Suppose that arbitrary delays are to be eliminated by setting $\Delta = 1$ as in Model #4. From Section 6.4.1 we know the reply telegram must arrive at least one cycle before the timer is stopped by the *FOP* data. There are at most six delay cycles to accommodate through the lossy link because the output buffer in East may be in the state $\mathsf{O}'(\mathtt{QZ})$ when protocol rule (1) (see Figure 6.5) is executed. The elapsed timer should therefore not timeout the remote route request before *seven* cycles have elapsed.

Adjusting Model #5 accordingly to lengthen the timeout, we can confidently address the model checker with the question

$$\mathsf{EastWest}^7_{\#5} \quad \models \quad \Xi \ ?$$

Intriguingly, this is false! There is a second failure mode revealed here which has been introduced by the duplication of telegrams—this turns out to be a reincarnation of the failure depicted in Figure 6.4. Observe that the SSI will normally send two reply telegrams (although it may receive two requests, at most one of these will succeed when $\Delta = 1$ because of the timer test in rule (3)). When the first acknowledgement is processed the elapsed timer will be stopped whether or not the route is actually locked (by rule (2)). However, before the second acknowledgement is returned the timer may well be restarted by another request for a cross-boundary route. As before, if the cancellation request precedes this the subsequent but spurious reply to the earlier remote route request may lock the route anyway. Ergo, $\mathsf{EastWest}^7_{\#5} \not\models \Xi$.

This problem may be somewhat artificial since it depends on the route first being set, and then quickly cancelled again:

$$\mathsf{EastWest}^7_{\#5} \quad \models \quad [\![\mathsf{qin}(\mathtt{QN})]\!](\nu Z.\neg\Theta \wedge [-\mathsf{qin}(\mathtt{QNX})]Z)$$

$$
\begin{array}{rcl}
\mathsf{H}(u) & \stackrel{\text{def}}{=} & \overline{\mathsf{get}}_H(u).\mathsf{H}(u) + \mathsf{put}_H(v).\mathsf{H}(v) \\
\mathsf{Control} & \stackrel{\text{def}}{=} & \mathsf{set}(\mathrm{Q}).(\mathbf{C}[\![\mathit{PRR}(\star\mathrm{Q})]\!]\mathsf{Control}) + \cdots + \mathsf{idl_in}(v).\mathsf{Filter}(v) \\
\mathsf{Filter}(v) & \stackrel{\text{def}}{=} & \mathsf{get}_H(u).\mathit{if}\ (v = u)\ \mathit{then}\ \overline{\mathsf{qin}}(0).\mathsf{Sync}\ \mathit{else}\ \overline{\mathsf{put}}_H(v).\overline{\mathsf{qin}}(v).\mathsf{Sync} \\
\mathsf{Sync} & \stackrel{\text{def}}{=} & \overline{\mathsf{sync}}.\overline{\mathsf{turn}}.\mathsf{Control} \\
\mathsf{SSI} & \stackrel{\text{def}}{=} & (\mathsf{Control} \mid \mathsf{Image} \mid \mathsf{Q}([\,]) \mid \mathsf{E}(0) \mid \mathsf{G}(0) \mid \mathsf{O}(0) \mid \mathsf{H}(0))\backslash L
\end{array}
$$

Model #6: Refining Model #5 to filter duplicate IDL inputs

where $\mathsf{qin}(\mathrm{QNX})$ is the (panel request) input to East that cancels the route. This property asserts that whenever the cross-boundary route is requested (from the initial state) $\neg\Theta$ remains invariant at least until such time as the route is subsequently cancelled. Apropos the possibility that the route is no longer available in the first Interlocking when the acknowledgement is received, the main reference [9] comments

> *this can only be due to failure, overrun, or signalman error.*

The first two possibilities (failure of the computer or signalling hardware, or a train overrunning a signal at red) can be reasonably ignored as these conditions will prevail long enough for *all* the reply telegrams to arrive, but we should ask whether operator error could lead to the failure the model suggests is inherent? It does not seem a very unlikely action on the part of the signal operator to set a route on the control panel and cancel it again, realising immediately that it was the wrong route to set, but the issue is difficult to resolve with certainty through informal argument—which is, in any case, hardly adequate when dealing with a system as complex as SSI.

It is not necessary to resort to informal argument however. Observe that once the first IDL message in a particular phase of the protocol has been *received*, any subsequent copies are redundant. This suggests that one need only record the identity of the *last* IDL input, and may discard the copies. This analysis leads to the final modification depicted in Model #6, where now $\{\mathsf{put}_H(v), \mathsf{get}_H(v)\} \subseteq L$. Here the queue is defined as before but a filter has been introduced. H is a new variable (a 'dummy telegram') that is not accessed from the Geographic Data but which the generic program uses to record the identity of the last IDL input from the other Interlocking (in general an array of these filter variables is needed). With this last change the model checker finally discharges the proof:

$$\mathsf{EastWest}^{7}_{\#6} \quad \models \quad \Xi$$

To summarise we have: instigated a one cycle timeout for queued IDL inputs, filtered the multiple copies of these inputs, passing only the first to the input buffer, and implemented a seven major cycle timeout for the elapsed timers used to protect IDL

communications on each side of the protocol. This completes the formal analysis of the model of the remote route request protocol.

6.5 Summary

In this chapter we have discussed the question of whether the remote route request protocol is intrinsically safe with respect to what it is designed to achieve, namely route *locking*. Informal analysis in Section 6.2 suggested that under unfavourable timing conditions failures may be observed that lead to states of the coupled system in which only the first half of a cross-boundary route is locked. These can arise because there is no means to guarantee the timely arrival of reply telegrams and, in particular, of discarding tardy replies from the other Interlocking. If the elapsed timer used to timeout remote route requests is subsequently restarted, the delayed reply telegram will be accepted as a bona fide acknowledgement to the earlier request, and the route locked accordingly. This is a cause for concern because it is not possible to predict what (safe) changes in the state of either Interlocking may have occurred in the interim.

At first sight there does not appear to be much of a 'protocol' here, at the level of Geographic Data, about which to reason formally. The control decisions—*i.e.*, about the messages that should be exchanged to invoke functions in the other SSI—are implemented in the Data, while the communications are handled by the subsystem over which they have no control. Yet there *is* a protocol that is revealed in the generic rules given on page 147 from which the route specific control data have been abstracted.

The generic rules in the Geographic Data form the basis of the formal model analysed through a succession of refinements in Section 6.4. Other components of the model represent features of the underlying communication mechanism—the input and output buffer processes, and the elapsed timers which are modelled using watchdogs. The weakest aspect of the model is its notion of time. The simple idea of counting major cycles can be justified on the one hand because the SSI updates the elapsed timers no more frequently than this, and on the other because the worst case is assumed where the major cycle is extended to its operational upper limit of one second. Of course, a less pessimistic, average case model should admit the worst case behaviour we have assumed (because the SSI does).

The first formulation, Model #3, confirms that there is a logical flaw in the remote route request protocol since this model assumes the IDL is a perfect communication medium. Model #4 demonstrates that as long as we can guarantee to deliver IDL telegrams within one major cycle, and service them within another, the protocol will function safely. This model illustrates that if buffered IDL inputs timeout after one major cycle, safety is guaranteed as the reply telegram always arrives, if it arrives at all,

at least one cycle before the elapsed timer reaches the timeout. The second refinement, in Section 6.4.2, shows that at the cost of implementing a priority queue (which may be a severe cost, in a real-time setting), longer servicing delays could be tolerated.

In Model #5 the assumption that the IDL functions perfectly was relaxed, and duplication of IDL telegrams was introduced to compensate for occasional losses. If the Interlockings only exchange messages once a major cycle we found that even with a one cycle timeout for buffered IDL inputs, the protocol cannot guarantee safety. The failure mode observed in Model #5 can be eliminated, however, by modifying the protocol slightly so the elapsed timers do not expire before seven major cycles have elapsed. This assertion is supported by the observation that

$$\mathsf{EastWest}^{7}_{\#5} \quad \models \quad [\![\mathsf{qin}(\mathtt{QN})]\!](\nu Z.\neg\Theta \wedge [-\mathsf{qin}(\mathtt{QNX})]Z)$$

holds in the initial state (in which all sub-routes are free, and routes unset). The property Θ is not invariant, however, since $\mathsf{EastWest}^{7}_{\#5} \not\models \nu Z.\neg\Theta \wedge [-]Z$. This, in turn, is because the duplication of IDL telegrams introduces a second failure mode.

It is interesting to note that it is the defensive measure employed to provide fault tolerance, the duplication of IDL telegrams, that leads here to a weaker safety argument. The interplay between safety requirements and fault tolerance is obviously delicate and this can lead, in the design of robust safety critical systems, to difficult compromises between the two. In this case, however, there seems no need to compromise safety in order to tolerate occasional lost IDL telegrams: Model #6 indicates that the simple strategy of filtering the redundant copies received eliminates the newly identified failure mode, whether or not it is ever likely to appear in practice. $\mathsf{EastWest}^{7}_{\#6} \models \nu Z.\neg\Theta \wedge [-]Z$ as long as queued IDL requests expire after a one major cycle delay.

While these results are reliable for the model described in here, it is important to realise that it is not fully general since only route locking in one direction over the boundary is examined, and since it also only captures the case where there are just two Interlockings connected via the internal data link (rather, two Interlockings that engage in locking any particular cross-boundary route). The model is in fact easy to generalise if one can achieve multi-way synchronisation on the IDL transmissions (*e.g.*, with SCCS, CSP, or perhaps more naturally with LOTOS [7] where one can model broadcast). In the next, concluding chapter we shall pick a number of concrete recommendations for the implementation of the remote route request protocol which the formal analysis in this chapter has brought forward. These, and the model described here and the proofs of its properties have already played an important rôle in the internal review which British Rail initiated in response to these discoveries in order to try and qualify the risk which may arise from the flaws discussed. The results of this assessment process are unfortunately not available at the present time.

Chapter 7

Safety in Interlocking Design

This final chapter summarises the main ideas discussed in the earlier chapters, and puts forward some conclusions and observations that are pertinent to the industrial usage of formal methods. Section 7.1 below considers the impact that implementing the changes to the remote route protocol suggested in the previous chapter may have on the system's overall performance. The discussion in Section 7.2 returns attention to the Geographic Data theorem prover described in Chapter 5: the data for the Leamington Spa pilot scheme [24] were tackled using the prototype with some success, but also some frustration since the sample data supplied to launch the project [66, 8] were far from representative. Even so, the results here are promising. Finally, Section 7.3 concludes this thesis.

7.1 Implementing Remote Route Locking Safely

The model and its formal analysis in the preceding chapter bring forward several concrete recommendations for the implementation of the remote route locking protocol. It is therefore important to consider the overall impact these might have when implemented in SSI. The elements we found to be missing were a second timer mechanism to ensure that messages cannot be delayed arbitrarily, and a filter mechanism to remove redundant copies of the reply telegrams. The reasons why these are important were discussed in depth in Section 6.4. The modifications to the SSI generic program their implementation entails are straightforward, but it is not our place here to argue that these *should* be implemented—that is for the appropriate signalling authority to decide after the results of their independent risk assessment procedures are finalised. The preceding analysis and the observations that follow were put forward as input to that process only:

Filter variables Introducing these involves a change to the SSI generic program, but not in the Geographic Data. In general, an array of these is needed, one for each (incoming) IDL telegram used to convey request codes. The required algorithm

need make at most two memory accesses: when a null input telegram is received the filter variable can be set to zero unconditionally; otherwise the filter variable needs comparing with the current input, and resetting accordingly (note that the test is data independent). It is important that receipt of a null telegram nevertheless causes the timestamps of all queued IDL messages to be adjusted ($0 \in \mathcal{I}$, in the definition of $\mathsf{A}(e, q)$ in Model #4). In the long run fewer requests will be queued and, hence, serviced.

Ring buffer The expiration of delayed IDL inputs requires a more complex algorithm and data structure. If ring buffer slots are two bytes wide one byte can hold the panel request code, the other an integral timestamp. On queuing an IDL input the timestamp in each non-empty buffer position requires adjusting. If an element in the queue expires the code can be set to null, and the algorithm to remove elements from the queue should discard null values. This slows the generic program, but only in minor cycles when the *IPT* data for IDL telegrams conveying request codes are processed. The main reference [9] indicates that these minor cycles are otherwise very lightweight, and the overhead will be very modest since the ring buffer has rather limited capacity. Note that changes to the generic program are localised in the modules that manage the queue of panel requests.

Deadlocks If the acknowledgement to a remote route request expires, the system may enter a state in which the tail half only of the route is set. In particular, the dummy outward sub-route will be free, so the (sub-route) release mechanism in the protocol cannot clear the tail portion of the route as it should. We may overcome this with a simple change in to the protocol to be implemented at the level of the Geographic Data:

$$\texttt{*QN if @CN},\, E = \texttt{stop then } Z_L^{out}\,\texttt{l},\, E = \texttt{0},\, G = \boxed{\texttt{QN}}\ \texttt{\textbackslash .}$$
$$\texttt{*AZ if } E = \texttt{stop},\, T_L^{out}\,\texttt{f then } Z_L^{out}\,\texttt{f \textbackslash .}$$

These should be compared with the rules (1) and (4) on page 147 (locking the dummy sub-route in rule (2) is now no longer necessary). With this change we find that:

$$\mathsf{EastWest}^7_{\#6} \quad \models \quad \nu Z.(\langle \mathsf{eobs}_{Z_L^{out}}(\texttt{f})\rangle tt \Rightarrow \langle \mathsf{wobs}_{Z_L^{in}}(\texttt{f})\rangle tt) \wedge [-]Z$$

The dummy inward sub-route is free in WEST whenever the dummy outward sub-route is free in EAST—a desirable property that the original specification does not enjoy. This change can never lead to an unwanted cancelling of a route in the other Interlocking.

1′. *QN if @CN, E =stop then Z_L^{out} l, E =0, G =QN \ .

2′. *AN if @CN, E <5 then @SN, T_L^{out} l \ E =stop .

3. *QM if @CM, E =stop then @SM, T_L^{in} l, Z_L^{in} l, E =3, G =AM \ .

4′. *AZ if E =stop, T_L^{out} f then Z_L^{out} f \ .

5. *QZ if E =stop then Z_L^{in} f, G =AZ \ .

6′. E =stop if E >7 \ .

7. if Z_L^{out} l, T_L^{out} f, E =stop then G =QZ \ .

8. T_L^{in} f if Z_L^{in} f, T_L c \ .

Figure 7.1: Modified rules for remote route locking and release

Elapsed timer Lengthening the timeout observed by the elapsed timers involves a change in the Geographic Data (in the *FOP* data file):

E =stop if E >7 \ .

Consequently when remote route requests fail there will be a (two second) longer pause before the telegram can be reused. (There will also be a longer pause when remote route requests succeed since the timer in WEST will take longer to stop.) Recall that the model counts major cycles, not seconds, and that the requirement is for a minimum of seven major cycles between starting the timer in EAST, and stopping it again through the above rule.

In Section 6.3.4 the protocol was axiomatised in terms of the eight generic rules in Figure 6.5. These, with the modifications suggested above, are reassembled in Figure 7.1. However, the urge to implement any changes to the SSI generic program should be suppressed unless a clear case can be made that the weaknesses identified in the protocol induce unacceptable risk. This requires signalling engineering judgement, not formal methods. However, while the delays needed to activate the fault (in its first incarnation) are hardly credible, an unfortunate combination of circumstances may lead to the tail part of a route being released prematurely: a long route straddling several SSI boundaries, a glitch on the IDL, a busy interlocking with a naturally long major cycle. Moreover, the second incarnation of the fault need not depend on poor timing properties at all—so it may in fact present the more serious hazard. That said, the signalling consequences of the tail portion of the route being free while the first part is locked need to be analysed in depth. At first sight, the scenario sketched in the discussion accompanying Figure 6.2 in Section 6.1 is pessimistic since if the dynamic signalling rules are implemented correctly the signal controlling the entrance to the route will be green for only a few seconds before being returned to red [67].

Guaranteeing safety is one thing, but we should also consider whether the performance of the SSI might be degraded in other ways if these changes are made. In fact, the practice of filtering *all* IDL inputs could lead to circumstances where the tail portion (in WEST) of a route remains locked even though it should have been released—*e.g.*, if the first cancellation request fails legitimately because the timer is running but all subsequent copies of the request are then filtered. Livelock in the protocol is undesirable as it would require the signal operator's intervention in order to 'unstick' the part of the route still locked in WEST (*e.g.*, by reselecting the route, and then cancelling it again). However, the formal model allows us to prove that the following filter is *sufficient* for safety:

$$\begin{aligned} \mathsf{Filter}(v) &\stackrel{\text{def}}{=} \mathsf{get}_H(u).\mathit{if}\ (v = u)\ \mathit{then}\ \mathsf{Sync}(0)\ \mathit{else}\ \mathsf{Sync}(v) \\ \mathsf{Sync}(v) &\stackrel{\text{def}}{=} \overline{\mathsf{put}}_H(v).\overline{\mathsf{qin}}(v).\overline{\mathsf{sync}}.\overline{\mathsf{turn}}.\mathsf{Control} \end{aligned}$$

The subtlety here is to filter only the second of any pair of successive, non-null telegrams, as long as the second is identical to the first of the pair. This is simpler than the version of the filter displayed in Model #6, and eliminates the possibility that the two Interlockings deadlock due to filtering all low priority IDL messages. Informally, this is because the protocol rule `*QZ` (in WEST) is guarded only by the timer test, and WEST cannot restart the timer except in response to a route request from EAST—in general though, WEST can restart its timer autonomously due to a remote route request in the opposite direction.

Finally, the elements of the model are assembled in the figure on page 161. In principle the Image component is given by a parallel composition of all the control variables defined in the SSI as specified in Section 3.2, but in practice the set is limited to just those that are needed to model the protocol (those explicitly mentioned in Figure 7.1). The sets $\mathcal{I}$, $\mathcal{O}$, and $\mathcal{Q}$ are as specified in Section 6.3.3, and define the interface to the IDL (through idl_in and $\overline{\mathsf{idl_out}}$) as well as the signal control panel: $\{\mathsf{qin}(v) \mid v \in \mathcal{Q} - \mathcal{I}\}$. The model analysed sets $M = 7$ and $\Delta = 1$ to achieve the final result, specialising the model to EAST and WEST which set routes in one direction only. It is thus not possible to claim that $M = 7$ achieves safe usage of the protocol when $\Delta = 1$ in general, although this is thought to be sufficient when only two Interlockings negotiate the locking of any single route. To answer the question formally, a more powerful model checker than that in the Concurrency Workbench is needed (although a more powerful machine may suffice in the binary case—the analysis used a 75 MHz Sun SparcStation 20 with 128M bytes of RAM, and it should perhaps be recorded tha none of the proofs involved took more than a few hours to complete).

$$\mathsf{E}(m) \stackrel{\text{def}}{=} \begin{cases} \overline{\mathsf{get}}_E(m).\mathsf{E}(m) + \mathsf{put}_E(n).\mathsf{E}(n) + \mathsf{stop}.\mathsf{E}(\mathtt{stop}) & \text{if } m = M \\ \overline{\mathsf{get}}_E(m).\mathsf{E}(m) + \mathsf{put}_E(n).\mathsf{E}(n) + \mathsf{sync}.\mathsf{E}(m+1) & \text{if } 0 \le m < M \\ \overline{\mathsf{get}}_E(m).\mathsf{E}(m) + \mathsf{put}_E(n).\mathsf{E}(n) + \mathsf{sync}.\mathsf{E}(m) & \text{if } m = \mathtt{stop} \end{cases}$$

$$\mathsf{G}(u) \stackrel{\text{def}}{=} \overline{\mathsf{get}}_G(u).\mathsf{G}(0) + \mathsf{put}_G(v).\mathsf{G}(v) \qquad u \in \mathcal{O}$$

$$\mathsf{Empty} \stackrel{\text{def}}{=} \mathsf{get}_G(v).\mathit{if}\ (v = 0)\ \mathit{then}\ \overline{\mathsf{idl_out}}(0).\mathsf{Empty}\ \mathit{else}\ \mathsf{O}'(v)$$

$$\mathsf{O}'(u) \stackrel{\text{def}}{=} \overline{\mathsf{idl_out}}(u).\mathsf{turn}.\mathsf{O}(u) + \overline{\mathsf{idl_out}}(0).\mathsf{turn}.\mathsf{O}(u) \qquad u \in \mathcal{O} - \{0\}$$

$$\mathsf{O}(u) \stackrel{\text{def}}{=} \overline{\mathsf{idl_out}}(u).\mathsf{turn}.\mathsf{Empty} + \overline{\mathsf{idl_out}}(0).\mathsf{turn}.\mathsf{Empty} \qquad u \in \mathcal{O} - \{0\}$$

$$\mathsf{Q}(q) \stackrel{\text{def}}{=} \begin{cases} \mathsf{qin}(e).\mathsf{Q}(app_l(e,q)) & \text{if } q = [\,] \\ \mathsf{qin}(e).\mathsf{A}(e,q) + \overline{\mathsf{set}}(e').\mathsf{Q}(q') & \text{if } q = [e'_n \mid q'] \end{cases}$$

$$\mathsf{A}(e,q) \stackrel{\text{def}}{=} \mathit{if}\ (e \in \mathcal{I})\ \mathit{then}\ \mathsf{Q}(app_l(e, age(q)))\ \mathit{else}\ \mathsf{Q}(app_l(e,q))$$

$$\mathsf{H}(u) \stackrel{\text{def}}{=} \overline{\mathsf{get}}_H(u).\mathsf{H}(u) + \mathsf{put}_H(v).\mathsf{H}(v) \qquad u \in \mathcal{I}$$

$$\mathsf{Filter}(u) \stackrel{\text{def}}{=} \mathsf{get}_H(v).\mathit{if}\ (v = u)\ \mathit{then}\ \mathsf{Sync}(0)\ \mathit{else}\ \mathsf{Sync}(v) \qquad u \in \mathcal{I}$$

$$\mathsf{Sync}(u) \stackrel{\text{def}}{=} \overline{\mathsf{put}}_H(u).\overline{\mathsf{qin}}(u).\overline{\mathsf{sync}}.\overline{\mathsf{turn}}.\mathsf{Control} \qquad u \in \mathcal{I}$$

$$\mathsf{Control} \stackrel{\text{def}}{=} \mathsf{set}(\mathrm{Q}).(\mathbf{C}[\![PRR(\star\mathrm{Q})]\!]\mathsf{Control}) + \cdots + \mathsf{idl_in}(v).\mathsf{Filter}(v)$$

$$\mathsf{SSI} \stackrel{\text{def}}{=} (\mathsf{Control} \mid \mathsf{Image} \mid \mathsf{Q}([\,]) \mid \mathsf{E}(0) \mid \mathsf{G}(0) \mid \mathsf{O}(0) \mid \mathsf{H}(0))\backslash L$$

$$app_l(e,q) = \begin{cases} q & \text{if } e = 0 \text{ or } length(q) = l \\ app(e,q) & \text{otherwise} \end{cases}$$

$$app(e,q) = \begin{cases} [idl_\Delta(e) \mid q] & \text{if } q = [\,] \\ [e'_n \mid app(e,q')] & \text{if } q = [e'_n \mid q'] \end{cases}$$

$$age(q) = \begin{cases} q & \text{if } q = [\,] \\ age(q') & \text{if } q = [e_n \mid q'] \text{ and } n = 1 \\ [e_{n \dot{-} 1} \mid age(q')] & \text{if } q = [e_n \mid q'] \text{ and } n \neq 1 \end{cases}$$

$$idl_\Delta(e) = \begin{cases} e_\Delta & \text{if } e \in \mathcal{I} \\ e_0 & \text{otherwise} \end{cases}$$

Figure 7.2: Final version of the Internal Data Link model in CCS

7.2 Leamington Spa

The Leamington Spa signalling scheme was the first in Britain to use Solid State Interlocking. The layout is moderately complex, covering about 15 track kilometers, with 48 track circuits, 15 sets of points, and 35 signals. The SSI needs 96 sub-routes (a further 14 sub-overlaps), but there are 71 routes (including main routes, warner routes, and call-on routes). Unfortunately no scheme plan for this interlocking was made available so the invariant had to be deduced from the data. Several track circuits had sub-routes defined through them only in one direction, and were therefore ignored, but several others had six or eight sub-routes and several MX4 terms were used to specify all mutually exclusive combinations. Then $|\mathbf{MX}| = 44$, $|\mathbf{PT}| = 30$, and $|\mathbf{RT}| = 137$ after simplifying and eliminating all inconsistent paths according to the algorithm discussed in Section 5.3.3. However, without reference to the scheme plan it is difficult to simplify the proof by the method implemented in Section 5.5.3 (which requires a given decomposition of **F**), so we proceed below without the aid of decomposition.

7.2.1 Strengthening the Invariant

The sub-route release rules are syntactically uniform, and easy enough to convert into HOL syntax. Taking the quadratic complexity measure claimed in Section 5.5 seriously, with the parameters defined above, we should expect the theorem prover to batch process the data in about two hours: in fact it took three and a half hours, which suggests coding inefficiencies in manipulating very large terms in HOL cannot be ignored. There were two rules rejected: one of these turned out to be due to a specification error (a spurious route variable was introduced in hand coding the invariant), but the other appears to be due to a genuine omission in the conditions for the rule. Without access to the scheme plan or control tables for the route(s) in question it is not possible to determine whether or not this is deliberate.

Only a handful of rules from the route request data were analysed. The first tried, `*QR41(3M)` in Figure 7.3, was selected due to its complexity. There are seven points along the route, and hence seven points tested in the condition. In Section 5.5 it was noted that each P_i `crf` (or `cnf`) test doubles the proof due to the implicit disjunction, making the efficiency of `PRR_TAC` rather poor. In fact the theorem prover could handle this rule without decomposing **F**—with modest space requirements (circa 30M bytes), but 'overnight' (circa 9 hours).

There is, however, a simple observation that allows us to sidestep the extraneous exponential complexity incurred through points "free to move" conditions: *if the points are controlled normal (respectively, reverse), they should be free to move normal (re-*

```
*QR41(3M) if R41(3M) a , P221 cfn , P222 cfr , P223 cfn ,
             P224 cfr , P225 cfr , P228 cfr , P231 cfn , U7DA f
          then R41(3M) s , S41 clear bpull , P221 cn , P222 cr , P223 cn,
               P224 cr , P225 cr , P228 cr , P231 cn , U7AD l , U8CD l ,
               U9AC l , U22BD l , U13AB l , U14AB l , U28AB l \.

*QR35(2M) if R35(2M) a , R35(2W) xs , P211 cfn , P213 cfn ,
             P224 cfn , U5BA f , U9BA f , @OL225A
          then R35(2M) s , S35 clear bpull , P211 cn , P213 cn , P224 cn,
               U2AB l , U3BC l , U4AC l , U5AB l , O8AD l , @OL225Q \.

/ OVERLAP AVAILABLE ( OUTER ROUTE SETTING )
*OL225A   if ( P225 cfn or P225 cfr ) \

/ CHOOSE & LOCK OVERLAP ( OUTER ROUTE SETTING )
*OL225Q (   if P225 cr  then O9AC l , O22BD l \
         or if P225 cfn then P225 cn , O9AB l \
         or                  P225 cr , O9AC l , O22BD l ) \
```

Figure 7.3: Sample *PRR* data from Leamington Spa

verse). This *property* of the data may be expressed in generic terms thus:

$$\mathbf{P} \stackrel{\text{def}}{=} (P\ \texttt{cn} \Rightarrow \mathit{PFM}(\texttt{*PN})) \wedge (P\ \texttt{cr} \Rightarrow \mathit{PFM}(\texttt{*PR}))$$

The informal notation to refer to the "free to move" data for the points in question was motivated in Section 2.4.2. Now, if **P** can be proved invariant, *i.e.*, $\{\mathbf{P}\}\ c\ \{\mathbf{P}\}$, this fact can be used in the proof that c leaves **F** invariant. This is because

$$(P\ \texttt{cn} \Rightarrow \mathit{PFM}(\texttt{*PN})) \Rightarrow (P\ \texttt{cn} \vee \mathit{PFM}(\texttt{*PN}) = \mathit{PFM}(\texttt{*PN}))$$

and similarly in the other points setting—since, in particular, $(p \Rightarrow q) \Rightarrow (p \vee q \Leftrightarrow q)$. To pursue this, we strengthen the invariant and prove $\{\mathbf{P}\wedge\mathbf{F}\}\ c\ \{\mathbf{P}\wedge\mathbf{F}\}$ since the desired rewrite rules (the antecedent in the above implication) can be deduced on-the-fly from the verification conditions which will now have the form: $? \vdash \mathbf{P}\wedge\mathbf{F}\wedge b \Rightarrow (\mathbf{P}\wedge\mathbf{F})[\tilde{v}/\tilde{x}]$. As before, b is the guard in the command c, if any. Although this introduces still greater complexity to the invariant, the proof becomes easier because the disjunctive term b can be simplified prior to the case analysis *on* b. For the rule cited earlier from the Leamington Spa scheme this strategy improves the performance twentyfold (28 minutes). Of course, failure to prove $\mathbf{P} \wedge \mathbf{F}$ invariant does not imply **F** is variant (the problem may be with **P**). However, we can check whether $\{\mathbf{P}\}\ c\ \{\mathbf{P}\}$ when the proof fails: fortunately this proof does not incur the same penalty in time complexity because the formula is much simpler than **F** (the same set of rewrite rules can be used to simplify b).

7.2.2 Swinging Overlaps

Dealing with extraneous complexity due to points tests in the panel route request data is one thing, but there also arise disjunctive tests in these data that cannot be avoided.

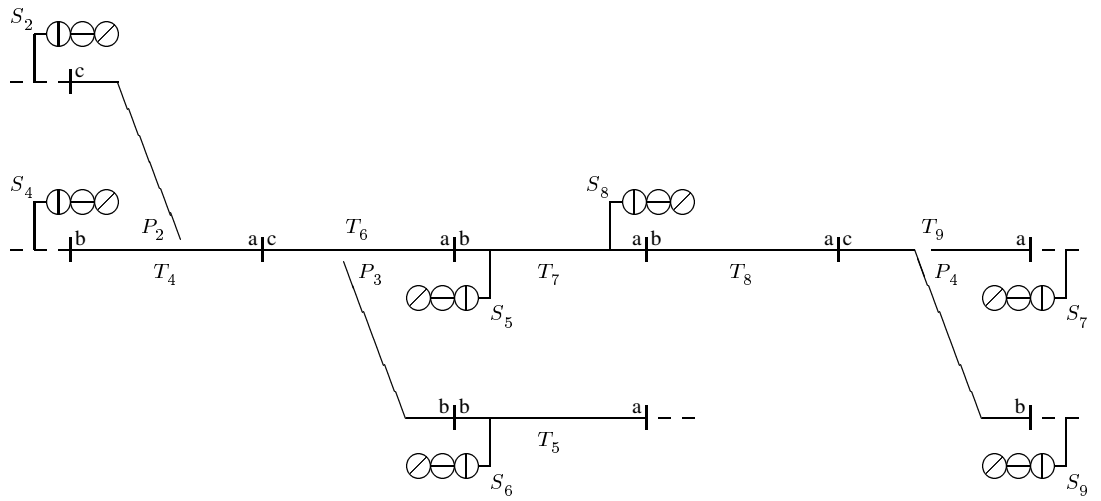


Figure 7.4: The overlap beyond S_5 for main routes up to this signal extends over the points P_3 normal (for a longer overlap the points P_2 can be either normal or reverse). Likewise for routes up to S_6, the overlap requires P_3 reverse. *Sub-overlaps* O_6^{ac}, O_6^{bc}, *etc.*, lock the overlap when the route up to the signal is locked.

The example to illustrate this point, which is the last we shall address here, concerns *overlaps*. There is often an overlap, or a choice of overlap, associated with the route(s) up to a signal that should be locked when the route is set. For routes terminating at S_8 for example (see Figure 7.4), there may be an overlap which extends some distance into the track section immediately beyond the signal—no overlap is needed if the signal is placed at a sufficient distance from the end of the track section. For routes terminating at S_5 there will be a choice of overlap if the points P_2 are close to the signal, since they are *facing* the direction of traffic up to the signal. Although not part of the route *per se*, the overlap is locked with it in order to protect other traffic against a train overrunning the signal at red.

The scheme in Figure 7.4, modelled on EAST-WEST with the points P_3 having been rotated, supplies the necessary intuition to understand the second Leamington Spa example. The route request data for R_{75} (say), and the points P_3 will be extended with tests on the sub-overlaps. Suppose that there are no overlaps required for routes up to the opposing signals (S_4 and S_2), then

```
*Q75 if  P4 crf , P3 cnf , T9ca f , T7ba f
         then  R75 s , P4 cr , P3 cn , T9ac l , T8ab l , T7ab l , O6ac l \ .
*P3N   T6 c , T6bc f , T6cb f , O6bc f \
*P3R   T6 c , T6ac f , T6ca f , O6ac f \
```

The (sub-)overlap release conditions will not be described here, but from these rules we may observe that while the overlap is locked (O_6^{ac} `l`) it will not be possible to lock

a route from S_6. Expressed as invariants, the overlap locking conditions are thus:

$$\mathbf{MX}[O_6^{ac}, O_6^{bc}] \ \wedge \ \mathbf{PT}_{\texttt{cn}}(P_3, [O_6^{bc}]) \ \wedge \ \mathbf{PT}_{\texttt{cr}}(P_3, [O_6^{ac}])$$

(recalling the macros in Section 3.3). Moreover, one should add $R_{75}\,\texttt{l} \Rightarrow O_6^{ac}\,\texttt{l}$, that is $\mathbf{RT}(R_{75}, [O_6^{ac}])$, since this will be specified in the control tables for the scheme.

Now `PRR_TAC` can deal with such simple circumstances where the route and its overlap are locked in a single action, but the second example cited earlier (the rule `*QR35(2M)`) extends over points that are facing the route's exit signal. With respect to the scheme in Figure 7.4 the corresponding interlocking logic is expressed thus:

$$\texttt{*Q75 if } P_4\,\texttt{crf}, P_3\,\texttt{cnf}, T_9^{ca}\,\texttt{f}, T_7^{ba}\,\texttt{f}, \texttt{@OLAP-A}$$
$$\texttt{then } R_{75}\,\texttt{s}, P_4\,\texttt{cr}, P_3\,\texttt{cn}, T_9^{ac}\,\texttt{l}, T_8^{ab}\,\texttt{l}, T_7^{ab}\,\texttt{l}, O_6^{ac}\,\texttt{l}, \texttt{@OLAP-L} \ \backslash .$$

$$\texttt{*OLAP-A if } (P_2\,\texttt{cnf or } P_2\,\texttt{crf}) \ \backslash$$
$$\texttt{*OLAP-L if } P_2\,\texttt{cr then } O_4^{ac}\,\texttt{l}$$
$$\texttt{else if } P_2\,\texttt{cnf then } P_2\,\texttt{cn}, O_4^{ab}\,\texttt{l} \texttt{ else } P_2\,\texttt{cr}, O_4^{ac}\,\texttt{l} \ \backslash\backslash\backslash$$

Here an *evaluation set* has been used to specify the conditions for the longer overlap which will be shared by several routes; similarly, the *execution set* locks one of the overlaps (with a preference for that over P_2 reverse). However, the "free to move" conditions for P_2 do not test the sub-overlaps (O_4^{ab} or O_4^{ac}) in this example because the overlap selected may have to be changed *after* the route is locked:

$$\texttt{*P2N } \ T_4\,\texttt{c}, T_4^{ac}\,\texttt{f}, T_4^{ca}\,\texttt{f} \ \backslash$$
$$\texttt{*P2R } \ T_4\,\texttt{c}, T_4^{ab}\,\texttt{f}, T_4^{ba}\,\texttt{f} \ \backslash$$

This can happen, say, because the normal overlap through these points was locked with the route, but at a later time an onward route from S_5 over these points in the reverse direction requires to be locked. The signalling action to achieve this is called *swinging the overlap*.

The **MX** and **PT** properties for these longer overlaps may be defined in analogy with the simpler example above and, modulo the discussion in Section 5.3.3 about diverging routes, we have in particular:

$$\mathbf{MX}[O_4^{ac}, O_4^{ab}] \ \wedge \ \ldots \ \wedge \ \mathbf{RT}(R_{75}, [O_6^{ac}, O_4^{ab}]) \ \wedge \ \mathbf{RT}(R_{75}, [O_6^{ac}, O_4^{ac}])$$

If **F** is extended in this way to **F′**, there are two problems which emerge in attempting to prove $\{\mathbf{F}'\}$ *PRR*$(\texttt{*Q75})\,\{\mathbf{F}'\}$. Before discussing these, first note that `@OLAP-L` is in sequence with the assignments made in this command, and that the overlap is consequently locked in two actions. The overall form of the term in the goal is that of a sequence nested in a conditional: $\texttt{if}(b_1, a_1\texttt{;}\,\texttt{ife}(b_2, a_2, \texttt{ife}(b_3, a_3, a_4)))$, which gives

rise to four nontrivial verification conditions (and one trivial one). Now this proof goal degenerates into the two subgoals

$$? \vdash \{\mathbf{F}'\}\ \texttt{if}\ P_4\ \texttt{crf}\ldots\ \texttt{then}\ldots\ O_6^{ac}\, l\, \{\mathbf{F}'\}$$
$$? \vdash \{\mathbf{F}'\}\ \mathit{PRR}(\texttt{*OLAP-L})\, \{\mathbf{F}'\}$$

because the heuristic implemented in `VC_TAC` (by `SEQ_TAC`, in Section 5.2.3) moves $\mathbf{F}'$ before the second `if` statement.

- However, it is not hard to see that the first of these subgoals cannot be proved since, in particular, $\mathbf{RT}(R_{75}, [O_6^{ac}, O_4^{ac}])$ (part of $\mathbf{F}'$) does not necessarily hold at the intermediate point. With some reprogramming we can circumvent this as follows:

  $$? \vdash \{\mathbf{F}'\}\ \texttt{if}\ P_4\ \texttt{crf}\ldots\ \texttt{then}\ldots\ O_6^{ac}\, l\, \{\mathbf{F} \wedge B\}$$
  $$? \vdash \{\mathbf{F} \wedge B\}\ \mathit{PRR}(\texttt{*OLAP-L})\, \{\mathbf{F}'\}$$

  where B carries through to the second step of the proof all of the information in the guard (and the command) in the first part—'$b_1 \wedge a_1$' in this instance. But it turns out that $\mathbf{F} \wedge B$ alone is, in general, too weak for the second part of the proof; the syntax-driven heuristic to counter this is fragile, but in this case drops $\mathbf{RT}(R_{75}, [O_6^{ac}, O_4^{ac}])$ and $\mathbf{RT}(R_{75}, [O_6^{ac}, O_4^{ab}])$ from $\mathbf{F}'$.

- The reasoning above would be adequate for the Leamington Spa data but for the second problem which is that $\{\mathbf{F}'\}\ \mathit{PRR}(\texttt{*OLAP-L})\, \{\mathbf{F}'\}$ cannot be proved. Ultimately, this is because $\mathbf{MX}[O_4^{ac}, O_4^{ab}]$ cannot be proved invariant (similarly $\mathbf{PT}_{\texttt{cn}}(P_2, [O_4^{ac}])$ and $\mathbf{PT}_{\texttt{cr}}(P_2, [O_4^{ab}])$). We can falsify $\mathbf{MX}[O_4^{ac}, O_4^{ab}]$ when O_4^{ac} `l` and the preconditions for R_{75} are satisfied, if $\mathit{PFM}(\texttt{*P2N})$ is satisfied, but P_2 are neither controlled normal, nor controlled reverse. The proof is possible if one assumes P_2 `cn` *or* P_2 `cr`, but in the absence of this constraint there is insufficient information in the "free to move" conditions for these points when the test is to select an appropriate overlap. Thus, the problem is that $\mathbf{MX}[O_4^{ac}, O_4^{ab}]$ cannot be proved invariant in principle, rather than just in practice, because these points are part of a swinging overlap.

However, although $\mathbf{MX}[O_6^{ac}, O_6^{bc}]$ *is* a safety critical property, it is apparent that $\mathbf{MX}[O_4^{ac}, O_4^{ab}]$ is not since these sub-overlaps do not have a (points) locking rôle—*neither sub-overlap through T_4 is tested in any geographic condition for the scheme in question.* One can reasonably ask, therefore, what is the purpose of the sub-overlap(s) through T_4? The answer is that they serve no purpose—and this is also true of `O22BD` (see Figure 7.3) in the Leamington Spa data. (However, these variables would have an interlocking function if, for instance, routes up to S_4 also required an overlap—the conditions for selecting such an overlap would need to check O_4^{ac} `f` , O_4^{ab} `f` before proceeding locking O_4^{ba}, say.)

In summary, therefore, the invariant for the Leamington Spa data is the conjunction $\mathbf{P} \wedge \mathbf{F}'$, where $\mathbf{F}'$ extends **F** with the **MX**, **PT**, and **RT** terms for the fixed overlaps, but only the **RT** terms for the swinging overlaps (there is only the one mentioned, in fact). Improving `VC_TAC` so that it pushes information forwards, the analysis proceeds slowly since no decomposition has been used. The rules cited in Figure 7.3 represent the most complex proofs conducted. The latter is the more computationally demanding because there are effectively four verification conditions to prove (these examples took approximately 28 minutes and 94 minutes respectively).

7.3 Conclusions

In summing up, there are several broad categories of observations that can be made here, or recalled from earlier summaries at the end of individual chapters. These concern *theorem proving* and theorem provers in general; the pivotal issue of *semantics* as a means to realising formal proofs of Geographic Data invariants; *model checking*, and the models we must build and validate when designing complex systems. There are naturally some ramifications for the railway signalling industry (in particular) which cannot be ignored. These are set out in the sections to follow.

7.3.1 Theorem Proving

Developing a theorem prover for Geographic Data from scratch is a major undertaking, and not one to be readily embarked upon by engineers in the railway signalling industry themselves. This is a task for mathematicians and computer scientists with a flair for formal logic, but such people are unlikely to produce tools useful to railway signalling engineers if, as one would expect, their work is conducted without due awareness of the practical problems faced in interlocking design. But the specialisation of an existing tool to a particular class of problems to solve is much less daunting; while still not easy perhaps, it gives one a real opportunity to carry out the technical development of the theorem prover in an environment that is informed by the engineering issues at hand. This approach to checking safety properties of geographic data has the merit that one quickly obtains a prototype with which to explore certain pertinent considerations—such as which properties can be proved, what representations of the data are appropriate, and which theorem proving strategies are likely to be fruitful.

Now the proof sketches in Chapter 4 notwithstanding, the discovery of the tactics needed to guide the theorem prover is very much an experimental process—not least because one needs to get a feeling for how the underlying algorithms massage the goal. Such feedback is of course invaluable, when the proof fails, because of the incompleteness of higher-order logic. But those same sketches (Propositions 4.4 and 4.5) are, it

goes almost without saying, very much more valuable to anyone who wishes to understand why $\vdash \{\mathbf{F}\}\, c\, \{\mathbf{F}\}$ is true: the sketch renders the proof intelligible (to the reader) in a way that the tactic can never do—since the reader, as opposed to the programmer, has not benefited from experimenting with the theorem prover to discover the tactic in the first place.

Yet is it not the theorem itself which is important? Indeed, but in fact there are three essential elements in our proof methodology for Geographic Data:

1. A proof sketch such as that for Proposition 4.5 which explains why **F** is invariant. This will be convincing evidence for the certifier, or other responsible party, who must ultimately pronounce the SSI installation 'safe'. Note, in passing, that this is a mathematical argument at the first level of Rushby's characterisation of formal methods (page 5, and [83, pages 15–21]).

2. A tactic—a proof script that is really no more than a sophisticated program—that guides the mechanical edition of the the invariance proof. There is no requirement that the tactic correspond exactly, or even approximately, to the informal sketch. (For the record, `SRR_TAC` was greatly influenced by the proof of Proposition 4.4, but `PRR_TAC` was hardly influenced at all by the proof of Proposition 4.5.)

3. The theorem $\vdash \{\mathbf{F}\}\, c\, \{\mathbf{F}\}$, for each c, and the formal proof (which shall remain inscrutable). This represents the *hard* evidence that c is safe with respect to **F**; the certifier need only verify that the formal proof exists.

Lest the reader be confused, it is worth pointing out that it is not necessary to provide a rigorous (level 1) argument to support every formal proof (*i.e.*, for each c); it is only important to select a representative c, as in Chapter 4, and to convincingly show why each property, like **MX**, is invariant.

The problem with this vision of adapting general purpose theorem provers like HOL to specific application domains, is that the tools do not tend to be optimised for non-interactive use. In particular, HOL suffers a well-known quadratic time penalty when translating between different representations of the terms of the logic; this penalty becomes debilitating when formulae become too large—it is difficult to be specific here, but terms the size of those needed in the safety proofs for THORNTON JN. are probably 'too large'. However, such term translations are not strictly required for non-interactive theorem proving. So, could we not then abandon the need to interact with the theorem prover, or abandon our adherence to HOL? As we saw in Chapter 5, the ultimate aim *is* to achieve full automation; however, as already emphasised, there are tangible gains to be had in active experimentation, particularly in the proof of concept phase of the development.

Switching to a more efficient platform is obviously appealing. PVS [77] is an attractive alternative because its designers have carefully integrated many decision procedures for arithmetic and first-order logic. These make swift work of what in HOL would be several minor proof steps, so the granularity of the (interactive) proof is somewhat coarser in PVS than it would be in HOL. Whether this has a dramatic effect on non-interactive efficiency remains to be established. However, a problem with PVS is that it is not an open programming system, and it is considerably more difficult to specialise the theorem prover to application specific task. Also, as a programmer, one has to be very cautious not to undermine the logical integrity of the proofs because, amongst other reasons, Lisp does not enjoy the type security of ML. This is not to say that PVS is poorly structured: on the contrary, it is extremely well designed, but being highly optimised for the convenience of the interactive user it is simply more difficult to modify as a result.

In the end, while theorists and their acolytes can argue the relative merits of fully expansive theorem provers like HOL, against agglomerations of decision procedures like PVS, such systems (or more realistically their commercial successors) will never be used in anger by industry if they are cumbersome, or inefficient. Let us therefore summarise the main point which is that

- flexibility and openness in interactive use, and
- efficiency in non-interactive symbol manipulation

are essential features of a theorem prover for higher-order logic that is to support application specific refinements in a convenient and practical way. If general purpose theorem provers like HOL succeed to a next generation, designers might like to reflect beforehand that their customers *will* tend to have specific applications in mind!

7.3.2 Semantics

The foundation of any formal approach to verifying properties of Geographic Data—even if *formal* is only taken to mean *machine checked*—is the semantics of GDL. One could argue, perhaps, that this language of sequential and conditional statements is too weak to make a big issue out of its formal definition, but this attitude is complacent at best. There are many reasons why formal semantics are important:

- they constitute a succinct and unambiguous reference for the language;
- being mathematically precise they redress the otherwise *ad hoc* character of its definition and natural language description;
- they provide a stable basis from which to extend (or even simplify) the language when the need arises, with consequences that are predictable;

- one can use the operational semantics to judge the correctness of the interpreter's implementation (*i.e.*, as a specification of its functionality);
- one can precisely identify the functionality to encode in the byte compiler (*e.g.*, whether it should only perform a one-to-one mapping at the level of syntax);
- and it becomes possible in principle, which would not otherwise be the case, to state and prove properties of GDL programs.

In Chapter 5 we used the formal (denotational) semantics to define our Geographic Data checker. But even if this particular verification approach is rejected, one could still use the operational semantics to define the finite state machine whose properties can be checked, with the additional reassurance that the control interpreter implements the very same machine—witness the construction in Chapter 3.

The focus on semantics raises the central question of their validity. Neither the denotational semantics in Chapter 5, nor the operational semantics in Chapter 2, claim to be definitive—the objective has been primarily to demonstrate how the semantics issue can be approached, and why it is important. However, while both accounts of the language are thought to be reasonable, their validity will still have to be established through experimentation—that is, essentially, through simulation—since they have been defined long after the language's original conception.

The question of validity can be approached in part formally by demonstrating a *correspondence theorem* to relate the operational and denotational semantics of GDL; one often wants to prove the semantics are in equivalence, but weaker relations like containment are also interesting. The absence of such a theorem from this thesis is perhaps an error of omission, although that was never one of its goals. In any case, much of the work needed has been done in the theorem (Theorem 2.1) at the end of Chapter 2. The missing step is to show that the operational interpretation of the Geographic Data with the map search converted to *if-then-else* normal form, coincides with the denotational interpretation of Section 5.2. Unfortunately, to prove equivalence, it will be necessary to give a more precise denotational account of the 'inversion' of the control bits when they are assigned in points memory—compare Section 5.3.2 with 2.4.2. The equivalence proof would provide indirect evidence that the semantics are valid, but at the very least it would demonstrate that they are mutually consistent.

Whether or not one proves that the operational and denotational semantics coincide, the question of validity remains. It is pertinent to ask, therefore, whether our HOL implementation of the Geographic Data theorem prover would be seriously compromised if the formal semantics did need to be revised? Well, if it turns out that one of the rules in Figure 5.1 is incorrect this may effect the derivation of a few of the underlying theorems, but only as far as `VC_TAC`. On the other hand, if it turns out to be unsatisfactory

to maintain the separation between the theory of the semantics of the language, from the (HOL) theory needed to represent datatypes like *points*, then the changes may be more far reaching (there again, perhaps not since specialising the assignment axiom to concrete datatypes at the outset is a rather trivial change). The HOL theory of the semantics of GDL is in fact quite robust and has already been through several painless iterations. Of course, establishing a completely different basis for the embedding, say via the operational semantics in the style of Camilleri and Melham [15], would be a more significant undertaking.

In summary then, raising the issue of the semantics of the Geographic Data Language at the outset helps to achieve the goal of providing a dedicated theorem prover for the language. It also establishes a proof methodology that is largely independent of SSI. This may be of considerable practical interest in the future since the next generation of solid state interlockings are already under development. The code (that is, the data) in the current SSI installations do not need to be discarded if their semantics can be preserved—and our proofs will of course remain valid. The corpus of Geographic Data should be reused if possible since it is an asset (rather than a legacy, or something to be ashamed of) if it has demonstrated that it is trustworthy through being used.

7.3.3 Model Checking

The question of validity arises again in the model developed through Chapters 3 and 4 (to examine safety properties of Geographic Data), and through Chapters 3 and 6 (to investigate the remote route request protocol). In the first case this is ameliorated to considerable extent by the fact that the construction in Section 3.2 is derived directly from the operational semantics set out in the preceding chapter. (In the interests of historical accuracy it is perhaps fair to note that the CCS construction in fact came first.) But the CCS model admits much more behaviour than does the SSI (with its data), so one still has to defend the choice of model. Well, the arguments put forward in Chapter 3 will not be rehashed here: it suffices to note that if **F** is invariant in the chaotic model, we can conclude that it will also be invariant in the more orderly environment of the railway—*given* that we accept the semantics, or translation, by which the model is derived. That is to say, the abstraction is conservative with respect to safety properties such as **F**. This 'trick' of abstraction, where one relaxes the environmental constraints that are present in the system one is modelling, is perhaps the most common modelling device to be found in the systems verification literature.

Our further development of the CCS model in Chapter 6 to investigate the remote route request protocol was a more conventional exercise in modelling and analysis. It is pertinent to ask, though, how the error in the design was found in the first place? From a methodological standpoint it might perhaps have been more satisfying to report that the

(first) flaw was discovered only after a suitable model of the inter-SSI communications had been built and probed for conformance to the stated requirements. However, the formal methods approach to system verification seldom works in quite this way. In fact, this design error was revealed during the *process of modelling the protocol*, as an indirect result of the deep introspection needed to write down the eight axioms in Figure 6.5. One can think of this loosely as 'during the specification stage' of the design of the process—since the analysis was conducted *post hoc*.

On the other hand, the second logical flaw in the protocol was indeed found during verification, and actually remained hidden—and unsuspected—until the supposed repaired model failed the safety proof. This experience very clearly illustrates how design dependability is at once improved by the attention to detail needed to produce a formal specification, and how one's confidence in the design is both deepened and justified by submitting the formal specification (the model) to the exacting analysis needed to prove that it meets its requirements. These activities encourage one to 'look in all the corners' in order to explicitly state one's assumptions about the environment in which the device is to operate, and to explore all of its possible behaviours within the confines of the model. There are both tangible and intangible results, or rewards, from pursuing these technical developments (which in our case of the remote route request protocol really did unfold in the manner described in Chapter 6). The tangible result is a greatly ruggedised description of the system design; the intangible results are present in our deeper understanding of the device to be built—its aims, how it works and, most importantly, why it functions as it does (safely, in the end).

As long as one has the patience to experiment a little in order to find the right level of abstraction, process calculus is an excellent mathematical formalism with which to model systems in general, but especially communication protocols. Being essentially algebraic in character, process calculi are much more than finite state machines with syntax. The subtle interplay between syntax and semantics (for CCS in particular) is perhaps the key which allows one to *present* what would otherwise be a monolithic state machine as a collection of cooperating agents. Such object-oriented decompositions, tightly bound to one's design intuitions, are a great boon for systems enginnering. Yet while building such a formal description of a complex system and formulating its mathematical properties is a highly skilled activity, checking that these properties hold of the model is best left to a machine. But a substantial amount of the skill (and energy) of the modeller is invested in expressing the model in such a way that the machine—in this case a model checker—can complete the task at all. This is not as it should be: tools such as the Concurrency Workbench should support, more than they dictate, the level of abstraction at which one operates.

It is not wholly fair to ridicule the CWB for its inefficiency in the face of chal-

lenging industrial problems. Explicit (graph) representations of finite state machines give one a rich choice of analysis methods, but one soon runs into the 'state space explosion' problem. With his *second* model checker [13], Clark handled this problem with seeming dexterity by representing the model symbolically instead: this innovation works well enough for hardware verification where states are bit vectors, but BDDs have yet to establish their utility in modelling software and systems in general. Nevertheless, Taubner *et al.* [32] at Siemens have, acknowledging the power of CCS as a modelling paradigm, developed successful (in-house) industrial prototypes having CWB functionality by exploiting symbolic representations of the transition relations. Commercially supported variations on the theme can be had in FDR (Formal Systems (Europe) Ltd.) for CSP, and (recently) in the LOTOS toolset.

There again, the CWB itself was never built as an industrial prototype for CCS, and it should not be mistaken for one even though there are now many case studies in the open literature where industrially relevant problems have been solved using the tool. What these studies show, and the analysis here of British Rail's remote route request protocol is hopefully a pedagogic example, is that for a successful application of CCS—and through incautious extrapolation, formal methods in general—calls for a careful balance between intellectual input and push-button verification. At present the balance may be weighted too heavily in the direction of intellectual effort. Yet despite industry's impatience, push-button verification may not be the panacea it at first seems. For no matter how efficient model or equivalence checkers become we shall always be able to conceive of designs that are too large to handle, and any tool's success, if measured in terms of state counts as is the current trend, will only encourage us to tackle more concrete design descriptions. However, when one is seeking safety assurance in interlocking design it cannot be ignored that it is through abstracting complex behaviour into simple models that one achieves the necessary understanding. This cannot be had simply by pushing buttons. Thus, there may be much that is positive in tools that are limited if they encourage abstraction in system design and analysis—though it is mildly embarrassing to hold up the CWB as an example of this!

7.3.4 Railway Signalling

Overall, the integrity of the interlocking depends on many distributed elements, and on the coordination of their activities through the protocols that connect them. Firstly, the integrity of each SSI is paramount: the Geographic Data that configure each installation have to demonstrate conformance to the prevailing principles of railway signalling, and that demonstration has to become an integral part of the design process. The approach to the problem sketched in this thesis is perhaps more sophisticated than that of other researchers in the field, but then the Geographic Data Language is markedly more

complex than the competitors we have encountered like STERNOL [88] or Vital Logic Code [37]. Secondly, the integrity of the generic architecture is also fundamental to the success of SSI: this has to be verified only once of course, but one has to take great care, and expend considerable effort, to identify the requirements *a priori*. For the remote route request protocol in particular, that processing all panel requests within one major cycle is in fact a *safety requirement* is certainly not obvious; that it also turns out not to be sufficient for safety illustrates the power of the modelling process, and the need to attempt formal proofs.

When viewed from the systems engineering perspective one would expect such analyses to be performed early in the software (and system) lifecycle, rather than after several years in service. This begs the question, therefore, of whether the analysis could be carried out by engineers in the railway signalling industry themselves? In principle, of course; but in practice the successful uptake of formal methods calls for a broadening of the skills base in the signalling community. However, as we have seen, the problems addressed by formal methods are not first and foremost those of signalling engineering (that is, of interlocking design), but rather those of computer systems engineering. Clearly, with the growing interest in computer based signalling systems the skills base in the railway signalling community is already broadening in this direction. And it is not unreasonable, though it may yet seem naïve, to expect the computer systems engineers who deign such safety critical systems to be fluent in formal methods. In the medium term the prospect for formal methods exploitation in interlocking design is therefore quite promising—particularly if one observes the close correlation between interlocking logic and mathematical logic (be it first-order logic or predicate calculus [88, 37], automata theory [48], or higher-order logic [99, 75]).

In the end, safety in interlocking design cannot be guaranteed by mathematical analysis alone—it would be unsafe to suppose that it could be. But since the railway is operated in the belief that adherence to the proper procedures introduces no undue risk, we have to assemble what evidence we can to justify that belief. This naturally calls for rigour throughout the design process. Since the control logic is now implemented in computer-based technology, particularly software, formal methods can bring unprecedented levels of confidence in the integrity of the design since formal proofs of critical properties provide the hardest possible evidence in support of safety assurance.

Bibliography

[1] F. Andersen, K. D. Pertersen, and J. S. Pettersson. Program verification using HOL-UNITY. In Joyce and Seger [50], pages 1–17.

[2] H. R. Andersen. Model checking and boolean graphs. In B. Krieg-Brückner, editor, *Proceedings of the 4th European Symposium on Programming (ESOP'92)*, volume 582 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[3] H. R. Andersen. *Verification of Temporal Properties of Concurrent Systems*. PhD thesis, rhus Universitet, 1993.

[4] W. Atkinson and J. Cunningham. Proving properties of a safety critical system. *Software Engineering Journal*, 6(2):41–50, 1991.

[5] A. Beveniste. Synchronous languages provide safety in reactive system design. *Control Engineering*, Sept. 1994.

[6] G. M. Birtwistle and P. A. Subrahmanyam, editors. *Current Trends in Hardware Verification and Automated Theorem Proving*. Springer-Verlag, 1989. Proceedings of the 1988 Banff Workshop on Hardware Verification.

[7] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems, North-Holland*, 14:25–59, 1987.

[8] A railway signalling case study for FOREST. Internal publication by British Rail Research, London Rd., Derby, England, 1988. Issue B, ELS-DOC-4314.

[9] SSI Data Preparation Guide. Published by British Railways Board, Feb. 1990. ELS-DOC-3080, Issue K of SSI8003-INT and supplements; British Rail Research, London Road, Derby, England.

[10] J. Bradfield and C. Stirling. Local model checking for infinite state spaces. *Theoretical Computer Science*, 96:157–174, 1992.

[11] J. C. Bradfield. *Verifying Temporal Properties of Systems with Applications to Petri Nets*. PhD thesis, University of Edinburgh, 1991. Available as CST-83-91.

[12] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–91, 1986.

[13] J. R. Burch, E. M. Clark, et al. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 428–39. Computer Society Press, 1990.

[14] J. R. Burch, E. M. Clark, et al. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 13(4):401–24, 1994.

[15] J. Camilleri and T. Melham. Reasoning with inductively defined relations in the HOL theorem prover. Technical Report 265, University of Cambridge Computer Laboratory, 1993.

[16] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[17] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(1):56–68, 1940.

[18] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–63, 1986.

[19] R. Cleaveland. Tableau-based model checking in the modal mu-calculus. *Acta Informatica*, 27:725–47, 1990.

[20] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based verification tool for finite-state systems. In *Proceedings of the 9th IFIP Symposium on Protocol Specification, Testing and Verification*, pages 287–302. North-Holland, 1989.

[21] G. V. Conroy and C. Pulley. Logical methods in the formal verification of safety-critical software. Presented at the IMA Conference on Dependable Computing, Sept. 1993.

[22] D. Craigen, S. Gerhart, and T. Ralston. *An International Survey of Industrial Applications of Formal Methods—Purpose, Approach, Analysis and Conclusions*, volume 1. NIST (National Institute of Standards and Technology), 1993.

[23] D. Craigen, S. Gerhart, and T. Ralston. *An International Survey of Industrial Applications of Formal Methods—Case Studies*, volume 2. NIST (National Institute of Standards and Technology), 1993.

[24] A. H. Cribbens. Solid State Interlocking (SSI): an integrated electronic signalling system for mainline railways. *Proc. IEE*, 134(3):148–58, 1987.

[25] A. H. Cribbens and I. H. Mitchell. The application of advanced computer techniques to the generation and checking of SSI data. *Proceedings of the Institute of Railway Signalling Engineers*, 1992.

[26] W. J. Cullyer and W. Wong. Application of formal methods to railway signalling—a case study. *IEE Computing and Control Engineering Journal*, 4(1):15–22, 1993.

[27] R. de Simone. Higher-level synchronising devices in Meije-SCCS. *Theoretical Computer Science*, 37:245–67, 1985.

[28] E. A. Emerson. Temporal and modal logic. In *Handbook ot Theoretical Computer Science* [96], chapter 16.

[29] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the modal mu-calculus. In *Proceedings of the 1st Annual IEEE Symposium on Logic in Computer Science*, pages 267–78. Computer Sociecty Press, 1986.

[30] R. Enders, T. Filkorn, and D. Tauber. Generating BDDs for symbolic model checking in CCS. In *Proceedings of the 3rd Workshop on Computer Aided Verification*, 1991.

[31] A. Fantechi, S. Gnesi, and G. Ristori. Model checking for action-based logics. *Formal Methods in System Design*, 4:187–203, 1994.

[32] S. Fisched, A. Scholz, and D. Taubner. Verification in process algebra of the distributed control of track vehicles—a case study. *Formal Methods in System Design*, 4(2):99–122, 1994.

[33] R. Floyd. Assigning meanings to programs. In *Mathematical Aspects of Computer Science*. American Mathematical Society, 1967.

[34] M. J. C. Gordon. HOL: A proof generating system for higher-order logic. In G. M. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, Kluwer International Series in Engineering and Computer Science, pages 73–128. Kluwer, Boston, 1988.

[35] M. J. C. Gordon. Mechanizing programming logics in higher-order logic. In Birtwistle and Subrahmanyam [6], pages 387–439.

[36] M. J. C. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[37] J. Groote, J. Koorn, and S. van Vlijmen. The safety guaranteeing system at station Hoorn-Kersenboogerd (extended abstract). In *Proceedings of the 10th Annual Conference on Computer Assurance (COMPASS'95)*, pages 57–68, Gaithersburg, Maryland, 1995.

[38] A. Gupta. Formal hardware verification methods: A survey. *Formal Methods in System Design*, 1:151–238, 1992.

[39] W. A. Halang and B. J. Krämer. Safety assurance in process control. *IEEE Software*, pages 61–7, Jan. 1994.

[40] J. Harrison. Binary decision diagrams as a HOL derived rule. *The Computer Journal*, 38(5), 1995.

[41] M. Hennessey. *Algebraic Theory of Processes*. Foundations of Computing. MIT Press, London, 1988.

[42] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer-Verlag, 1980.

[43] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[44] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10), 1969.

[45] C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice Hall, 1985.

[46] Software for computers in the application of industrial safety-related systems. International Electrotechnical Commission, 1994. IEC Standard 1131, Part 3.

[47] M. Ingleby. A Galois theory of local reasoning in control systems with compositionality. Presented at the IMA Conference on Dependable Computing, Sept. 1993.

[48] M. Ingleby and I. Mitchell. Proving safety of a railway signalling system incorporating geographic data. In *Proceedings of SAFECOMP'92*, pages 129–134. IFAC, Pergamon Press, 1992.

[49] A. Jowett. *Jowett's Railway Atlas of Great Britain and Ireland*. Patrick Stephens Ltd. (Haynes), 1989. ISBN: 1-85260-086-1.

[50] J. Joyce and C. Seger, editors. *Higher Order Logic Theorem Proving and its Applications*, volume 780 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[51] T. King. Formalising British Rail's signalling rules. In *Proceedings FME'94: Industrial Benefit of Formal Methods*, volume 873 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[52] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–54, 1983.

[53] L. Lamport. The temporal logic of actions. Technical Report 79, Digital Systems Research Center, 1991.

[54] K. G. Larsen. Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72:265–88, 1990.

[55] S. Malik. Analysis of cyclic combinational circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 13(7), 1994.

[56] K. Mark Hansen. Formalising railway interlocking systems. Technical report, Department of Computer Science, Technical University of Denmark, July 1994. Presented at the Nordic Seminar on Dependable Computing Systems.

[57] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon, 1992.

[58] T. F. Melham. Automating recursive type definitions in higher order logic. In Birtwistle and Subrahmanyam [6], pages 341–86. Proceedings of the 1988 Banff Workshop on Hardware Verification.

[59] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[60] R. Milner. The use of machines to assist in rigorous proof. *Phil. Trans. R. Soc. Lond.*, 312:411–22, 1984.

[61] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, 1990.

[62] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1990.

[63] R. Milner. Interpreting one concurrent calculus in another. *Theoretical Computer Science*, 75:3–13, 1990.

[64] R. Milner. Operational and algebraic semantics of concurrent processes. In *Handbook ot Theoretical Computer Science* [96], chapter 19.

[65] R. Milner and M. Tofte. Co-induction in relational semantics. *Theoretical Computer Science*, 87:209–22, 1990.

[66] I. H. Mitchell. Proposal for an SSI data checking tool. Internal publication by British Rail Research, London Rd., Derby. DE2 8YB, June 1990.

[67] I. H. Mitchell, Nov. 1995. Personal communication.

[68] The procurement of safety critical software in defence equipment (Guidance). UK Ministry of Defence, Apr. 1991. (Interim) Defence Standard 00-55, Part 1.

[69] The procurement of safety critical software in defence equipment (Requirements). UK Ministry of Defence, Apr. 1991. (Interim) Defence Standard 00-55, Part 2.

[70] Hazard analysis and safety classification of the computer and programmable electronic system elements of defence equipment. UK Ministry of Defence, Apr. 1991. (Interim) Defence Standard 00-56.

[71] F. Moller and C. Tofts. A temporal calculus of communicating systems. In *Proceedings of CONCUR'90*, volume 458 of *Lecture Notes in Computer Science*, pages 401–15, 1990.

[72] M. J. Morley. An heuristic approach to state space reduction of communicating parallel systems. Master's thesis, University of Edinburgh, 1989. Summarised in [73].

[73] M. J. Morley. Tactics for state space reduction on the CWB. Technical Report ECS-LFCS-90-109, Laboratory for Foundations of Computer Science, University of Edinburgh, 1990.

[74] M. J. Morley. Modelling British Rail's interlocking logic: Geographic data correctness. Technical Report ECS-LFCS-91-186, Laboratory for Foundations of Computer Science, University of Edinburgh, 1991.

[75] M. J. Morley. Safety in railway signalling data: A behavioural analysis. In Joyce and Seger [50], pages 465–74.

[76] O. S. Nock. *Railway Signalling—a treatise on the recent practice of British Railways*. Adam and Charles Black, London, 1980. Prepared under the direction of a committee of the Institution of Railway Signalling Engineers under the general direction of O. S. Nock.

[77] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–52. Springer-Verlag, 1992.

[78] D. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[79] L. C. Paulson. *The Isabelle Reference Manual*. Computer Laboratory, University of Cambridge, 1993.

[80] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends. In *Current trends in Concurrency*, volume 224 of *Lecture Notes in Computer Science*, pages 510–84. Springer-Verlag, 1986.

[81] V. Pratt. A decidable μ-calculus. In *Proceedings of the 22nd Annual IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1981.

[82] G. M. Reed and A. Roscoe. A timed model of communication sequential processes. In *Proceedings of ICALP'86*, volume 226 of *Lecture Notes in Computer Science*, pages 314–23. Springer-Verlag, 1986.

[83] J. Rushby. Formal methods in the certification of critical systems. Technical Report CSL-93-7, SRI International, 1993. Also: NASA CR 4551 *Formal Methods and Digital Systems Validation for Airborne Systems*.

[84] B. Sanders. Eliminating the Substitution Axiom from UNITY logic. *Formal Aspects of Computing*, 3(2):189–285, 1991.

[85] R. C. Short. Software validation for a railway signalling system. In *Proceedings of SAFECOMP'83*, pages 183–193. IFAC, Pergamon Press, 1983.

[86] K. Slind. AC unification in hol90. In Joyce and Seger [50], pages 437–49.

[87] G. Stlmarck. A proof theoretic concept of tautological hardness. Incomplete manuscript circulated to interested parties for review., May 1994.

[88] G. Stlmarck and M. Säflund. Modelling and verifying systems and software in propositional logic. In *Proceedings of SAFECOMP'90*, pages 31–6. IFAC, Pergamon Press, 1990.

[89] C. Stirling. Modal and temporal logics for processes. Technical Report ECS-LFCS-92-221, Laboratory for Foundations of Computer Science, University of Edinburgh, 1992. Lecture notes for the *4th European Summer School in Logic, Language and Information*, University of Essex.

[90] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In *Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT'89)*, volume 351 of *Lecture Notes in Computer Science*, pages 368–82. Springer-Verlag, 1989.

[91] C. Stirling and D. Walker. A general tableau technique for verifying temporal properties of concurrent programs. In *Proceedings of the International BCS-FACS Workshop on Semantics for Concurrency*, Workshops in Computing, pages 1–15, Berlin, 1990. Springer-Verlag.

[92] D. Syme. A new interface for HOL - ideas, issues and implementation. In T. E. Schubert and P. J. Windley, editors, *Higher Order Logic Theorem Proving and its Applications*, volume 971 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[93] R. D. Tennent. *Semantics of Programming Languages*. International Series in Computer Science. Prentice Hall International, 1991.

[94] L. Thèry, Y. Bertot, and G. Kahn. Real theorem provers deserve real user interfaces. In *Proceedings of the fifth ACM SIGSOFT Symposium on Software Development Environments*, volume 17 of *Software Engineering Notes*, 1992.

[95] C. Tofts. *Proof Systems and Pragmatics for Parallel Programming*. PhD thesis, University of Edinburgh, 1990.

[96] J. van Leeuwen. *Handbook ot Theoretical Computer Science*, volume B. Elsevier, 1990.

[97] D. Walker. Automated analysis of mutual exclusion algorithms using CCS. *Formal Aspects of Computing*, 1:273–92, 1989.

[98] P. J. Windley. The `records` library. Unpublished, although available on-line., 1993. `http://lal.cs.byu.edu/lal/holdoc/library/-records/records.html`.

[99] W. Wong. *A Formal Theory of Railway Track Networks in Higher-order Logic and its Applications in Interlocking Design*. PhD thesis, University of Warwick, 1991.

[100] W. Wong. Modelling bit vectors in HOL: the `word` library. In Joyce and Seger [50], pages 371–84.

Appendix A

Glossaries

The three glossaries in the pages that follow are intended to aid the reader by filling out some of the details omitted from the general discussion in the Introduction, and in Chapter 2. The entries are grouped according to the following conventions:

A.1 Glossary of railway signalling terminology, describing the main components of the railway and the control system;

A.2 Glossary of Solid State Interlocking terminology, describing the central notions that are mainly specific to this kind of signal control system;

A.3 Glossary of Geographic Data terms, describing the organisation and purpose of the entities in the geographic database.

The reader may benefit simply from reading these pages through in a linear fashion on the first occasion.

A.1 Glossary of Railway Signalling Terms

APPROACH LOCKING The aspect displayed by a signal depends on the tracks in advance (they should be clear of traffic, and all points should be locked, before a green aspect is displayed) *and* in the rear. For example, a signal at green should not be reset to red if there is a train within sighting distance of the signal—otherwise the driver may be unable to halt the train before reaching the signal. Approach locking is the interlocking function that maintains the displayed aspect while a train is approaching a signal. See ***ROUTE RELEASE*** (A.1).

ASPECT Traffic indication displayed by a signal. For semaphore signals this is indicated by the position of the arm—vertical (and thus not visible) meaning proceed, horizontal (and often accompanied by a red warning lamp) meaning halt. An intermediate diagonal position was introduced to permit traffic to proceed with caution (and expect to halt at the next signal). Modern power lamp signals often

have four aspects: red, yellow, double yellow, and green. Intermediate flashing yellow aspects introduce further speed indications. See also ***SIGNAL*** (A.1).

BERTH TRACK SECTION This is the track section in which a train will be standing when facing a signal at red. By analogy with a ship's berth in harbour, it is the 'berth' for all routes onward from the signal.

CONTROL PANEL A visual display at the signal control centre indicating the current status of railway network. A schematic plan of the railway will be illuminated to indicate which routes are set and to show the current position of trains. Other indications display the on/off state of signals, and sometimes the position in which points switches are detected. Modern technology has introduced video display units to replace 'mosaic' control panels, but the principles of operation are the same. Operators issue commands at the panel to reconfigure the network and route trains to their destinations. Points can be moved independently by moving a points *key switch* on the panel; routes are set by pressing *buttons* at the entrance and exit signals (in that order), and released by pulling the entry signal button. See ***PANEL REQUESTS*** (A.3) and ***ROUTE SETTING*** (A.1).

CONTROL TABLE Control tables list the locking conditions for each route in the interlocking. Each route has an entry that specifies which track circuits must be clear before the entry signal can be turned off (including overlaps, and overlaps associated with conflicting routes). The control table also specifies the required orientation of the points along the route, and specifies which signals control access to conflicting routes (which should be on before the route is set). Some railway authorities and providers of signalling equipment have abandoned the control table as the means of specifying routes—due to the difficulty of verifying that the route locking conditions are adequate.

CROSSOVER A configuration of points that permit trains to cross between parallel tracks. The points at either end of the crossover are usually coupled together so that they may only move in unison. See ***POINT SWITCH*** (A.1).

DIAMOND CROSSING A section of track where two lines cross without the possibility of allowing traffic to switch between lines. The area marked out between the four rails is trapezoidal, hence the name!

INTERLOCKING As a noun, a generic name for the signal control system as a whole. Interlockings may be of several types: ground-frame interlockings operated mechanically, relay-based systems with electromechanical controls, and

computer controlled or 'solid state' interlockings. When capitalised, Interlocking always abbreviates *Solid State Interlocking* in the main text.

INTERLOCKING LOGIC A term used for the logical relationships between physical entities in the railway such as points, signals, track circuits, and so on. In SSI, this is programmed in the Geographic Data; in relay-based interlockings this is hardwired into the relay circuitry, and in ground-frame interlockings it is manifest in the mechanical linkages between physical components.

LAMP PROVING (CIRCUIT) A separate *proving circuit* is built into power lamp signals to check that the selected aspect is in fact drawing current—this will be the case unless both the main and auxiliary filaments in the lamp are broken. Lamp proving therefore offers a positive indication that the selected signal aspect is being displayed.

OVERLAP Main signals (as opposed to repeaters, *etc.*.) act as the exit signal for routes up to the signal, and the entrance signal for all onward routes. The overlap track section (and circuit) lies immediately beyond the signal. The overlap is not strictly part of the route up to the signal, but while the signal is on it should be kept clear of other traffic to afford protection against a train inadvertently overrunning the signal. The overlap track circuit is distinct from the (full) track circuit in advance of the signal. See also ***SWINGING OVERLAPS*** (A.1).

POINTS KEY This is a switch on the signal control panel that allows the signal operator to lock the points semi-permanently in normal or reverse position. The centre setting for the points key releases control of the points to the interlocking. See also ***POINTS MEMORY*** (A.3), ***POINTS DATA*** (A.3).

POINT (SWITCH) Points are mechanical devices in the railway to change the path that trains may take through a junction. The switch positions are called *normal* and *reverse* respectively, the former usually referring to the mainline, the latter to the branch. An electrical contact is used to detect—*i.e.*, to give a positive indication—that the points are lying in the position to which they have been called by the interlocking. When a route is set the points along it will be locked (logically, but also physically clamped) to prevent their being moved again before the train has passed.

ROUTE Routes are definite paths between pairs of signals—at least on British railways, other railway authorities often define routes in different ways. *Main* routes are defined between consecutive pairs of main signals; *warner* routes coincide

with main routes, but permit traffic to proceed only under caution (*i.e.*, the entrance signal will typically not display a green aspect); *call-on* routes have a very specific function: to allow an engine to be coupled to a train—since to achieve this one must violate the safety principle that only one train may occupy a track section at once.

ROUTE LOCKING/SETTING Before a signal can be turned off, an onward route must be set. In the first instance, this involves checking that the availability conditions for the route in question are met—*e.g.*, to check that no conflicting route is currently set; then the route must be *locked* so that subsequent actions taken by the interlocking do not change these conditions. Secondly, the route must be *proved*—the control system expects a positive indication that the points along the route are detected in the required positions, for example. Finally, the entrance signal can be turned off, but the aspect displayed will depend on the class of route, the aspect displayed by the next signal, and other factors. See ***PANEL REQUEST*** (A.2), ***ROUTE RELEASE*** (A.1) and ***ROUTE MEMORY*** (A.3).

ROUTE RELEASE Under normal conditions a route, having been set, will be released automatically once a train passes the entrance signal. This switches the signal back on. As it proceeds the route is released behind the train—*e.g.*, once it is clear of a set of points they can be unlocked and subsequently moved in setting another, previously conflicting route. Otherwise, a route may be cancelled by the signal operator (usually in order to set an alternative route) but then the approach locking conditions must be met. See ***APPROACH LOCKING*** (A.1), ***SUB-ROUTE RELEASE*** (A.3).

SCHEME PLAN A scheme plan is a detailed drawing of the railway layout, in a diagrammatic form, that identifies all of the physical components of the interlocking. In particular all signals, points, track sections and track circuits will appear on the plan. Train control tables and Geographic Data are derived from the scheme plan. See also ***CONTROL TABLE*** (A.1).

SEMAPHORE Early type of signalling device, beloved of enthusiasts. The arm on the signal post is operated against a heavy counterweight so that effort is required to lift the arm to the vertical position. Should the mechanical linkage between signalbox and signal break, the weight will drop the arm to the horizontal position. By convention the horizontal position means stop! The semaphore is a simple, gravity operated, fail-safe device.

SIGNAL Signals control the linear movements of trains, and can give a speed indication to drivers by displaying one of a variety of aspects. A signal is *on* when it displays the red aspect, meaning halt; it is *off* otherwise, giving drivers permission to enter the track section ahead. Signals themselves may serve a variety of purposes: main signals for normal traffic control; route (or junction) indicators may warn drivers to slow down due to a diversion ahead; shunt and subsidiary signals have specialised functions in closely monitored situations. Signals are capable of displaying multiple aspects: two-aspect main signals will display either red or green aspects; two-aspect repeaters (intermediate signals between the entrance and exit signals) will normally display yellow or green aspects, but not red. Most modern signal installations on mainline railways use three or four aspect colour signals, with flashing yellow aspects for finer speed control.

SWINGING OVERLAPS If there are facing points in the overlap track section there may be a choice of overlap. For route(s) up to the signal it may not matter which overlap is selected in setting the route, but routes in the network beyond the signal may be unavailable because they conflict with the chosen overlap. Under careful control it is possible to swing the overlap—that is, to select another one—some time *after* the route has been set. Swinging overlaps is not an inherently safe activity (some railway authorities have outlawed the practice!) because this involves releasing the first overlap before setting the second. In particular, the points in the overlap will be 'undetected' whilst they are moved from one position to the other, and consequently the signal should come on (display the red aspect), but this would be unsafe if a train where within sighting distance.

TRACK CIRCUIT The track circuit is the primary safety device in the railway. Track circuits are always identified with a track sections, though there may be several electrically isolated track circuits in a single track section in a complex network. A track circuit is used to detect the presence of a train in the section. A voltage is applied across the rails, which may be detected to indicate that the section is *clear*. When a train is present the voltage between the rails drops due to the short circuit, and this registers the section *occupied* at the control centre. Track circuits fail on the safe side since a faulty circuit will indicate the presence of a train.

TRACK SECTION An identified section of the railway line that is controlled by a signal. The primitive components (segments, or parts) from which track sections are assembled are points, diamond crossings, and plain track. Track sections are electrically isolated from one another.

A.2 Glossary of SSI Terminology

(BASEBAND) DATA HIGHWAY The data highway is a bidirectional communications link between the central interlocking processor and the track-side functional modules. The data highway is operated at the rate of 20 k bits per second and uses a screened twisted-pair, duplicated for reasons of fault tolerance.

CENTRAL INTERLOCKING PROCESSOR The interlocking processor is mainly responsible for the safe operation of the railway network. This is usually referred to as *the* SSI in the main text (occasionally Interlocking, but then always capitalised, in the interests of avoiding terminological monotony). The central interlocking processors operate in (repairable) triple modular redundancy to achieve high levels of hardware reliability, and to afford fault tolerance. Each submodule is identical, running identical software and having identical copies of the Geographic Data, but independent RAM devices. See also ***GEOGRAPHIC DATA*** (A.2).

COMMAND TELEGRAM Command telegrams convey signalling controls to the equipment at the track-side. Eight control bits are bundled together with sender and receiver address and diagnostic data with five parity bits to form a truncated (31,26) Hamming code which is transmitted in Manchester encoded bipolar form, adding a second layer of error protection. The eight command bits are set up by commands in the Geographic Data. See ***OUTPUT TELEGRAM DATA*** (A.3).

CONTROL INTERPRETER The SSI is a data-driven control system. In this thesis, the control interpreter (often, just 'the control') is the name given to the generic software running in the SSI, sometimes referred to as the 'interlocking functional program' by other authors. This software interprets the Geographic Data, and it is this behaviour of the program that is of most interest in this thesis. The control interpreter has other functions, but all interlocking functions are encoded in the data except for a few very simple operations 'hardwired' into the interpreter for the sake of efficiency. See also ***INTERLOCKING FUNCTIONAL PROGRAM*** (A.2), ***GEOGRAPHIC DATA*** (A.3) and the discussion in Sections 1.3.2 and 1.4.

DATA TELEGRAM The Reply telegrams from track-side equipment to the SSI are encoded according to the same format as command telegrams. Data telegrams relay the inputs from detection devices in the track-side equipment to the central interlocking (lamp proving, points detection and track circuit inputs, for example). These inputs are typically copied directly to the internal state. See ***INPUT TELEGRAM DATA*** (A.3).

GEOGRAPHIC DATA These data specify the logical relationships between the components of the railway, encoding the signal control functions of the Interlocking. Stored in EPROM (60 k bytes of which are allocated, 20 k bytes of these required to hold the generic SSI software) the Geographic Data configure each SSI installation. Data and program together achieve the required signalling function—setting a route, releasing an overlap, and so on—but the data themselves can be considered a program that operates on a state that is composed of the collection of all control variables defined for the interlocking (one for each point switch, track circuit, *etc.*). See ***INTERNAL STATE*** (A.2) and ***CONTROL INTERPRETER*** (A.2), and Appendix A.3 where different classes of data are described.

GEOGRAPHIC DATA LANGUAGE (GDL) is a specialised design notation used by signal engineers to encode the interlocking logic. This simple language of assignment, sequence and conditional statements is general enough to code all signalling functions, but it is enriched by 'specials' designed to shorten the minor cycle execution time. Specials are directives to the interpreter to carry out simple functions efficiently—such as copying an input telegram bit to memory, for example. See ***SPECIALS*** (A.3), and Section 2.3.

INTERLOCKING FUNCTIONAL PROGRAM While its main function is that of interpreting Geographic Data, the generic SSI software also: initiates all communications with track-side functional modules; encodes and decodes all outgoing and incoming telegram data; performs single fault recovery; implements the TMR voting mechanism and shutdown procedure; implements the inter-SSI communications protocol; interfaces with the panel and diagnostic processors, and implements all startup routines. The interlocking functional program occupies about 20 k bytes of EPROM. The program is referred to as the *control interpreter* throughout the main text.

INTERNAL DATA LINK (IDL) The internal data link is a separate communications channel to provide inter-SSI communications. There will usually be more than one SSI at a single control centre, 30 of which may be connected to one IDL, but the current technology is limited so that an SSI can send (and receive) only up to 15 eight-bit messages. The IDL is primarily used for setting routes across SSI boundaries, and for controlling signals or points in the fringe area.

IDL TELEGRAM Telegrams sent over the internal data link convey two kinds of information. When used to carry status information between the two interlockings, each bit in the telegram is interpreted individually—like data telegrams received

over the baseband data highway. In these circumstances the individual bits are used to set up dummy signal or track circuit memories in the receiving interlocking. The other use for IDL telegrams is to carry *request codes*, as part of the remote route request protocol. The eight-bit telegram is interpreted as an integral request code which causes the receiving SSI to execute a specific interlocking function from the *PRR* data file. IDL telegrams can serve one, and only one, of these two purposes. See ***PANEL REQUEST*** (A.2), and ***REMOTE ROUTE REQUEST*** (A.2).

INTERNAL STATE The internal state of the SSI represents the current status of the railway—in the main text this is usually referred to as the *image* of the railway. A collection of control variables are defined and held in RAM: up to 256 track circuit memories are allocated, with 64 points and 128 signals, together with logical control variables for routes, timers, sub-routes, and other binary flags. These data represent 1,216 bytes of 'live' memory upon which the Geographic Data and control interpreter operate.

MAJOR CYCLE One major cycle is 64 minor cycles. A maximum of 63 TFMs may be attached to each SSI, the zeroth minor cycle being used for diagnostic purposes and updating the SSI with commands from the technician's console. A major cycle is 64 minor cycles in duration irrespective of the actual number of TFMs attached, with a lower limit of 608 ms, and an upper limit that should not exceed 1,000 ms. During a major cycle all flag operations data will have been processed once, as will all input and output telegram data, and all timers will have been adjusted once. Timers are only accurate to ± 2 s, and cannot be updated more than once a major cycle.

MINOR CYCLE The minor cycle is the basic execution cycle during which the SSI will process and issue one command telegram, and receive and process one reply telegram (from the TFM addressed in the previous minor cycle). Other required activities during the minor cycle include the processing of $1/64^{\text{th}}$ of the commands in the *FOP* data file, and updating $1/64^{\text{th}}$ of the approach locking, track circuit and elapsed timers in the interlocking. If these actions can be completed in under 9.5 ms the SSI will process one panel request, if any are pending. The minor cycle has a minimum duration of 9.5 ms, and should be no longer than 30 ms otherwise track-side modules may interpret the gaps in the communications as failures of the baseband data highway and enter the failure mode of operation.

MODE 1/2/3 STARTUP A 'mode 1' (2 or 3) startup is chosen by heuristics in the initialisation software. A 'mode 1' startup is the most severe, necessitating a reset of the entire contents of RAM: all bits are cleared to zero except the technician's controls and the elapsed timers whose contents are set to one. This initial state means that all routes are *unset*, all sub-routes and sub-overlaps are *locked*, and all timers are *stopped*; also, all technician's controls are applied, points are neither controlled normal nor reverse, and track circuits are undefined. Moreover the processing of panel requests is suspended while the system is brought up-to-date by incoming data telegrams, and while technician's controls are released manually from the technician's console. A 'mode 2' startup involves a similar reset, but preserves the technician's controls, and the system restarts automatically after a four minute suspension in processing panel requests. A 'mode 3' startup also preserves the status of route memory, and allows an immediate restart. See also ***TECHNICIAN'S CONSOLE*** (A.2), ***INTERNAL STATE*** (A.2), and Appendix A.3.

PANEL PROCESSOR The panel processor handles non-critical duties such as handling commands issued at the control panel (or automatic route setting computer) and passing them over to the interlocking processor, and updating the display. Panel processors are operated in duplex 'hot standby'.

PANEL REQUEST Signalling commands issued at the signal control panel are either route requests, route cancellation requests, or panel key requests (to move points 'manually'). The panel processor converts these into a stream of inputs to the SSI—but because both panel processors are normally operational, the SSI receives and executes two copies of each request. These are stored by the central interlocking in a ring buffer of bounded size, and processed during minor cycles which are otherwise completed in under the minimum minor cycle time. At most one panel request will be served in any minor cycle. See also ***MINOR CYCLE*** (A.2) and ***ROUTE REQUEST DATA*** (A.3).

REMOTE ROUTE REQUEST Routes that straddle interlocking boundaries require special treatment since two (or more) Interlockings must cooperate to set them up safely. When the Interlocking controlling the entrance signal receives a panel request for such a route, it issues a remote route request via the internal data link to the Interlocking controlling the tail portion of the route. Only if an acknowledgement to this remote request is received from the other Interlocking (within a prescribed period of delay) will the first Interlocking go ahead and lock the route. See Section 1.4.

TECHNICIAN'S CONSOLE The technician's console allows close monitoring of the internal state of several Interlockings at a signal control centre, and the on-line diagnosis of faults in the signalling equipment, *etc.*. The technician's console also allows one to impose (temporary) restrictions on the behaviour of the interlocking, by applying so-called *technician's controls*. These can be applied to routes (so that they are unavailable, and requests for them always fail), to track circuits (so they always appear occupied, irrespective of the actual state), to points (so they can be disabled in either the normal or reverse position), and to signals (to override the lamp-proving input from the TFM). Of these, only the 'availability bit' in route memory is accessible from the Geographic Data—so that an alternative route can be selected perhaps.

TRACK-SIDE FUNCTIONAL MODULE (TFM) These devices interface with the track-side signalling equipment. Two types of module are provided: one to drive signal aspects and detect lamp proving inputs, *etc.*; the other to drive points and detect their position contacts. Either type of module can report track circuit inputs. Both signal and points modules have identical interfaces to the baseband data highway, and are configured to respond to a command telegram with an immediate reply (data) telegram. Track-side functional modules provide power switching under duplicated microprocessor control—duplication here, as elsewhere in SSI, being designed to mask single faults and to drive the outputs to a safe state when unrecoverable faults are detected.

A.3 Glossary of Geographic Data Terminology

(ELAPSED) TIMER 64 bytes of RAM are reserved for 64 timers which may be used for any purpose in the Geographic Data—but they are usually associated with communications with other interlockings and swinging overlaps. Timers count seconds, to an accuracy of ± 2 s, upwards from zero to the 'sticking' value of 254. Timers are 'stopped' by setting their contents to 255: elapsed timers are stopped and started from the Geographic Data, but incremented by the control interpreter at most once a major cycle.

EVALUATION SET An evaluation set is a labelled block of tests on data variables which may be referenced in any context where a test is valid (but reference and label must be in the same data file). See also ***SPECIALS*** (A.3).

EXECUTION SET An execution set is a labelled block of arbitrary conditional code which may be referenced in any context where a command is valid (but reference and label must be in the same data file). See also ***SPECIALS*** (A.3).

FLAG MEMORY 128 bytes of RAM are allocated to flags (single bit variables). Flags include sub-routes and sub-overlaps whose states may be *locked* and *free*, and general purpose latches.

FLAG OPERATIONS DATA Each command in the flag operations data file (*FOP* data) is executed once a major cycle. One release rule is needed for each sub-route and sub-overlap, but any other data that require to be executed once a major cycle can be placed here.

INPUT TELEGRAM DATA One block of data is associated with each input telegram received from the track-side functional modules (in the *IPT* data file). The SSI is configured so that the input telegram processed in minor cycle m is the reply from the module addressed with a command telegram in cycle $m-1$ (modulo 64). Input telegram data update the detection bits in the image of the railway. *IPT* data are also specified for each telegram received over the IDL, and in the special case that these convey request codes the interpreter is configured to queue the appropriate 'panel' request.

MAP SEARCH Map searches (in the *MAP* data file) are frequently used to decide if route release conditions are met. A map search involves a look back from a feature reference (a signal or track circuit) for evidence of an approaching train (*i.e.*, an occupied track circuit).

OUTPUT TELEGRAM DATA The most complex interlocking logic is located in the *OPT* data file. One block of data is needed for each TFM addressed by the interlocking: data for points modules are simple (one just needs to drive the points to the position of the `c` bit in points memory) but signal aspects are interlocked with those of other nearby signals and track circuits, so setting the correct command bits in the output telegram requires a longer sequence of commands. *OPT* data are also needed for telegrams used to convey signal control data over the IDL.

PANEL REQUEST Each input from the signal control panel corresponds to a command to be executed from the panel (route) request data (*PRR* data file). These data list all route requests that arrive via the IDL or from the panel processor, and all route release requests. Points 'key' requests allow the operator to move points independently of setting a route over them. ***PANEL PROCESSOR*** (A.2) and ***ROUTE REQUEST DATA*** (A.3).

POINTS DATA Points "free to move" data (*PFM* data file) specify the conditions under which points may be switched, with one set of data required for each lie of

the points. *PFM* data may be called from other data files, particularly the *PRR* data in deciding route availability.

POINTS MEMORY 64 bytes of RAM are allocated to points memories, each of which contains two four-bit records (for the *normal* and *reverse* lie of the points). The 'controlled', 'detected', and 'key switch' fields of each record are under Geographic Data control, the fourth is used to disable the points and is only accessed by the program (technician's control).

ROUTE MEMORY 64 bytes of RAM are allocated to 256 route memories. Routes may be *set* or *unset*, this field being under the control of the Geographic Data: the 'available' flag is used to disable a route and is only testable.

ROUTE REQUEST DATA The *PRR* data file contains commands that are executed only on demand, when the SSI serves a panel request. Route request data specify the availability conditions, and locking conditions for each route defined in the Interlocking. Availability conditions need to check that points along the route can be moved to the required positions, and whether an opposing route is already locked—normally, it suffices to test the opposite sub-route to the first sub-route on the route in question, and the last sub-route on any directly opposing routes. The points "free to move" data (*PFM* data file) for each set of points on the route should cause the route request to fail if any route is locked over the points in the wrong direction. See also ***POINTS DATA*** (A.3).

SIGNAL MEMORY 128 signal memories are allocated, each requiring 3 bytes. Each signal memory includes an 'approach locking timer' (one byte), an aspect code (three bit), and a several other control flags for deciding which aspect to display, for sequencing the distant signals, and for deciding when the signal can be turned on, and the forward route(s) released.

SUB-ROUTE One sub-route is allocated to each path through a track circuit that lies on a route (so one sub-route may be part of several routes). Similarly, a *sub-overlap* is allocated for each path through an overlap track circuit that is part of an overlap. Sub-routes and sub-overlaps are boolean flags that may be *locked* or *free*.

SUB-ROUTE RELEASE DATA These data are located in the *FOP* data file, and specify the conditions under which sub-routes (and sub-overlaps) can be released. Usually, the first sub-route on a route requires the route *unset*, and the first track circuit *clear*; subsequent sub-routes are 'chained', requiring the previous sub-route(s) *free* and the track circuit *clear*.

SPECIALS Specials are directives in the Geographic Data Language that instruct the control interpreter to take short cuts in processing frequently occurring constructs. The volume of data, especially in *PRR* and *OPT* data files, can be reduced by putting common code in an evaluation set: the `@` special causes the interpreter to jump to the reference. Other specials are associated with input telegrams—typically to abbreviate the actions of testing a telegram bit, and setting the corresponding memory bit appropriately. The logic that the specials abbreviate can always be expressed in the conditional language.

TRACK CIRCUIT MEMORY 512 bytes of RAM are allocated for 256 track circuit memories. Each track circuit may be *clear* or *occupied*. Two single bit fields are used to give this indication, and three successive 'track circuit clear' inputs must be received before the *clear* field is set. Each record includes an eight bit timer to record how long the track circuit has been in the current state. The Geographic Data can test the timer along with the status flags – often used for automatic signals which revert to green after a suitable interval since the last train went through (automatic signals do not have routes associated with them in the same way as the fixed block main signals described in the main text).

Appendix B

Theory

In this technical appendix we summarise, rather briefly, the syntax and semantics of the mathematical formalisms used throughout the main text. There is no need to be comprehensive here since the theories used are by now well established and understood. This appendix is provided as a convenience, key references being cited below. Here we discuss the main notions underlying CCS, which formed the basis of the models developed in Chapters 3 and 6, and the modal μ-calculus and an associated proof system which we used in Chapters 4 and 6. Lastly, the proof tactics described in Chapter 5 are assembled in Appendix B.3.

B.1 Calculus of Communicating Systems

The Calculus of Communicating Systems is an an algebraic theory intended to describe communication between, and computations of, abstract *processes*. The theory developed in [61] took *observation equivalence* as the basis of deciding when two processes are to all visible intents, equivalent. The theory of observation equivalence was refined by Park [78] who introduced the *bisimulation* proof technique which is now fundamental to the theory. The equational theory of CCS is formulated on a refined notion of observation equivalence, that of observation congruence [64]. In [62] Milner has brought the theory up to date, the exposition being based around a wealth of examples.

CCS is not unique in what it sets out to achieve: it has spawned numerous *process algebras* that develop the theory in different directions. Of special note is Hoare's CSP which was developed independently of CCS [45]. CSP also takes as primitive the idea of indivisible action representing communication, but differs slightly in the semantics of the interaction between processes, as well as in its notion of equivalence.

$\dfrac{}{\alpha.P \xrightarrow{\alpha} P}$	$\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$	$\dfrac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$
$\dfrac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$	$\dfrac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}$	$\dfrac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$
$\dfrac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$	$\dfrac{P \xrightarrow{\alpha} P' \quad \alpha, \overline{\alpha} \notin L}{P\backslash L \xrightarrow{\alpha} P'\backslash L}$	$\dfrac{P \xrightarrow{\alpha} P' \quad A \stackrel{\text{def}}{=} P}{A \xrightarrow{\alpha} P'}$

Figure B.1: Transition rules for pure CCS

PURE CCS

In CCS, computation and communication are both abstractly represented by *actions*. Let $\mathcal{A} = \Lambda \uplus \{\tau\}$ be a set of actions, τ being a distinguished so-called 'silent' action. Λ is a set of *labels* having two disjoint subsets: Λ^+ is a set of *names*, and Λ^- is a set of *co-names*. We let $a, b, c \ldots$ range over names, $\overline{a}, \overline{b}, \overline{c}, \ldots$ range over co-names, and α, β range over $\mathcal{A}$. If $l \in \Lambda$ is a label, then its inverse is $\overline{l} \in \Lambda$, and for any label $\overline{\overline{l}} = l$. From this alphabet terms, or *agent expressions* are constructed according to the syntax:

$$P ::= \mathbf{0} \quad \Big| \quad \alpha.P \quad \Big| \quad P + P \quad \Big| \quad P \mid P \quad \Big| \quad P\backslash L \quad \Big| \quad P[f]$$

Informally, $\mathbf{0}$ represents a stopped or deadlocked computation; $\alpha.P$ can perform action α and will then behave as P; $P + Q$ represents choice—the agent can evolve either as P or as Q; $P \mid Q$ represents the parallel interleaving of the actions of P and Q; $P\backslash L$ is restricted in its visible behaviour—P cannot communicate via actions in the set L; finally, $P[f]$ behaves just as P, but the actions are renamed according to the bijection $f : \Lambda \to \Lambda$. Relabelling functions have the property that $f(\overline{a}) = \overline{f(a)}$; we can extend the domain to $\mathcal{A}$, but insist $f(\tau) = \tau$ for all f.

More formally, the semantics of CCS terms are given with respect to a *labelled transition system*: $(\mathcal{P}, \mathcal{A}, \{ \xrightarrow{\alpha} \mid \alpha \in \mathcal{A}\})$. $\mathcal{A}$ is a set of actions, $\mathcal{P}$ is a universal set of processes (or states), and for each $\alpha \in \mathcal{A}$, $\xrightarrow{\alpha} \subseteq \mathcal{P} \times \mathcal{P}$ is a labelled transition relation. The transitional semantics are then specified by the rules in Figure B.1. The last of these rules introduces the principle of definition: if A is defined to be P then A behaves exactly as P does. Definitions like $A \stackrel{\text{def}}{=} P$ are a means to introduce non-finite behaviour to CCS terms (because the *constant* A may appear in the body P). Another way is to explicitly introduce recursive terms: $\mathit{Fix}\,(X.\,P)$ where X may appear free in P. The rule for *Fix* is

$$\frac{P\{\mathit{Fix}\,(X.\,P)/X\} \xrightarrow{\alpha} P'}{\mathit{Fix}\,(X.\,P) \xrightarrow{\alpha} P'}$$

The two recursive forms are equivalent: the latter is more satisfactory in proofs, the former much more convenient in specifying models.

VALUE PASSING

Pure CCS describes only synchronisations between abstract agents. Whether one thinks of $\overline{a}.P$ as output and $a.P$ as input, or the other way around, is immaterial. The value passing calculus has a richer syntax since we think of values being communicated over channels, but it turns out to be no more expressive than the pure calculus. If we let V be some domain of values, the parameterised action $\overline{a}(e)$ represents the communication of the value $v \in V$ of the expression e over the channel a. On the other hand, $a(x)$ will bind the value received on a to the (value) variable x.

The syntax and semantics of the value passing calculus are given in the following translation table:

$$
\begin{array}{rcl}
[\![a(x).P]\!] & = & \sum_{v \in V} a_v.[\![P\{v/x\}]\!] \\
[\![\overline{a}(e).P]\!] & = & \overline{a}_e.[\![P]\!] \\
[\![\sum_{i \in I} P_i]\!] & = & \sum_{i \in I} [\![P_i]\!] \\
[\![P \mid Q]\!] & = & [\![P]\!] \mid [\![Q]\!] \\
[\![P \backslash L]\!] & = & [\![P]\!]\{l_v \mid l \in A, v \in V\} \\
[\![P[f]]\!] & = & [\![P]\!][\hat{f}] \quad \text{where } \hat{f}(l_v) = f(l)_v \\
[\![\mathit{if}\ (b)\ \mathit{then}\ P]\!] & = & \begin{cases} [\![P]\!] & \text{if } b = \mathit{true} \\ \mathbf{0} & \text{otherwise} \end{cases} \\
[\![A(\tilde{e})]\!] & = & A_{\tilde{e}}
\end{array}
$$

In translating $a(x)$ therefore, a distinct action a_v is created for each value $v \in V$ the parameter x may take. The output $\overline{a}(e)$ also becomes an indexed action—this time the complement to a_e will be a member of $\{a_v \mid v \in V\}$. Note that the indexed sum $\sum_{i \in I} P_i$ generalises the binary sum given earlier. The index set may be finite or infinite. We may also admit the notation $\prod_{i \in I} P_i$ as long as I is a finite set, but note that synchronisations between parallel agents are always pair-wise, by the third rule for composition.

$A(\tilde{e})$ denotes a parameterised agent, $A(\tilde{x}) \stackrel{\text{def}}{=} P$ being the appropriate definition. Only the value variables $\tilde{x}$ may appear free in P. The two-armed conditional is defined in terms of the simpler guarded command:

$$(\mathit{if}\ (b)\ \mathit{then}\ P) + (\mathit{if}\ (\neg b)\ \mathit{then}\ Q)$$

Some examples should help to clarify the translation mechanism.

Example B.1 Recall the definition of a register, from Chapter 3:

$$\mathsf{Reg}(x) \stackrel{\text{def}}{=} \overline{\mathsf{get}}(x).\mathsf{Reg}(x) + \mathsf{put}(y).\mathsf{Reg}(y)$$

If we suppose that the values are binary, $\mathcal{B} = \{0, 1\}$, this definition will be translated into an indexed set of pure CCS definitions:

$$\{\mathsf{Reg}_v \stackrel{\text{def}}{=} \overline{\mathsf{get}}_v.\mathsf{Reg}_v + \textstyle\sum_{u\in\mathcal{B}} \mathsf{put}_u.\mathsf{Reg}_u \mid v \in \mathcal{B}\}$$

Hence:

$$\begin{aligned}\mathsf{Reg}_0 &\stackrel{\text{def}}{=} \overline{\mathsf{get}}_0.\mathsf{Reg}_0 + \mathsf{put}_0.\mathsf{Reg}_0 + \mathsf{put}_1.\mathsf{Reg}_1\\ \mathsf{Reg}_1 &\stackrel{\text{def}}{=} \overline{\mathsf{get}}_1.\mathsf{Reg}_1 + \mathsf{put}_0.\mathsf{Reg}_0 + \mathsf{put}_1.\mathsf{Reg}_1\end{aligned}$$

♣

Example B.2 For a second example, recall that in translating Geographic Data into CCS in Chapter 3 we obtained a term like:

$$\mathsf{get}_P(v).\mathit{if}\ (v = \mathtt{cn})\ \mathit{then}\ P\ \mathit{else}\ Q$$

for suitable (unparameterised) P and Q. See Figure 3.4 on page 55, for instance. The data domain for points we considered there was just $\{\mathtt{cn}, \mathtt{cr}\}$. Then:

$$\begin{aligned}[\![\mathsf{get}_P(v).\mathit{if}\ (v = \mathtt{cn})\ \mathit{then}\ P\ \mathit{else}\ Q]\!] &= \mathsf{get}_{P_{\mathtt{cn}}}.[\![\mathit{if}\ (\mathtt{cn} = \mathtt{cn})\ \mathit{then}\ P\ \mathit{else}\ Q]\!]\\ &+ \mathsf{get}_{P_{\mathtt{cr}}}.[\![\mathit{if}\ (\mathtt{cr} = \mathtt{cn})\ \mathit{then}\ P\ \mathit{else}\ Q]\!]\\ &= \mathsf{get}_{P_{\mathtt{cn}}}.[\![P]\!] + \mathsf{get}_{P_{\mathtt{cr}}}.[\![Q]\!]\end{aligned}$$

which is just $\mathsf{get}_{P_{\mathtt{cn}}}.P + \mathsf{get}_{P_{\mathtt{cr}}}.Q$ since P and Q are simple constants. In the main body of the thesis this term is written $\mathsf{get}_P(\mathtt{cn}).P + \mathsf{get}_P(\mathtt{cr}).Q$ simply to avoid an unreadable profusion of subscripts. ♣

BISIMULATION

Of central importance to the development of the theory of CCS is the notion of *bisimulation*. Intuitively, two agents or states in a transition system, are bisimilar if each can simulate the other—that is, every action of one agent is matched by some action of the other in such a way that the resulting states are also bisimilar. This is succinctly captured by the following:

Definition B.1 A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ is a **strong bisimulation** if $P\mathcal{S}Q$ implies for all $\alpha \in \mathcal{A}$:

1. if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\alpha} Q'$ with $P'\mathcal{S}Q'$ for some Q', and
2. if $Q \xrightarrow{\alpha} Q'$ then $P \xrightarrow{\alpha} P'$ with $P'\mathcal{S}Q'$ for some P'.

We say that two agents P and Q are **strongly bisimilar**, written $P \sim Q$, if $P\mathcal{S}Q$ for some strong bisimulation $\mathcal{S}$. □

$P \sim Q$ is considered a strong equivalence relation (a congruence with respect to the operators of the language), as it gives no special status to the silent action τ. A coarser equivalence, more useful in practice, is obtained by considering only the visible behaviour of an agent. We may define an *observation* using the transitive closure of the τ relation: $P(\stackrel{\tau}{\longrightarrow})^* P'$, which is normally written $P \Longrightarrow P'$. Then for $\alpha \in \mathcal{A}$, $P \stackrel{\alpha}{\Longrightarrow} P'$ is defined by composing $\stackrel{\alpha}{\longrightarrow}$ on the left and on the right by $\Longrightarrow$: $P \Longrightarrow \stackrel{\alpha}{\longrightarrow} \Longrightarrow P'$. Finally we define the visible content of the observation by $P \stackrel{\widehat{\alpha}}{\Longrightarrow} P'$, which denotes $P \Longrightarrow P'$ when $\alpha = \tau$, and $P \stackrel{\alpha}{\Longrightarrow} P'$ otherwise. This leads to:

Definition B.2 A binary relation $\mathcal{W} \subseteq \mathcal{P} \times \mathcal{P}$ is a **weak bisimulation** if $P \mathcal{W} Q$ implies for all $\alpha \in \mathcal{A}$:

1. if $P \stackrel{\alpha}{\longrightarrow} P'$ then $Q \stackrel{\widehat{\alpha}}{\Longrightarrow} Q'$ with $P' \mathcal{W} Q'$ for some Q', and
2. if $Q \stackrel{\alpha}{\longrightarrow} Q'$ then $P \stackrel{\widehat{\alpha}}{\Longrightarrow} P'$ with $P' \mathcal{W} Q'$ for some P'.

Two agents P and Q are **observation equivalent**, written $P \approx Q$, if $P \mathcal{W} Q$ for some weak bisimulation $\mathcal{W}$. □

From the definition it follows, therefore, that in order to prove two agents P and Q are equivalent it is enough to find a bisimulation containing the pair.

Example B.3 By way of an example the relation $\{(a.\mathbf{0}, \tau.a.\mathbf{0})\} \bigcup \mathbf{Id}$, where **Id** is the identity relation, is a weak bisimulation containing the agents $a.\mathbf{0}$ and $\tau.a.\mathbf{0}$. However, this example also serves to show that observation equivalence is not a congruence relation: $b.\mathbf{0} + a.\mathbf{0} \not\approx b.\mathbf{0} + \tau.a.\mathbf{0}$. ♣

Although observation equivalence is not fully substitutive, it almost is. In fact, only the preemptive power of the τ action in the context of sum, as above, breaks the congruence. This leads to the following formulation of the definition of *equality* between agents:

Definition B.3 P and Q are **observation congruent**, written $P = Q$, if for all $\alpha \in \mathcal{A}$:

1. if $P \stackrel{\alpha}{\longrightarrow} P'$ then $Q \stackrel{\alpha}{\Longrightarrow} Q'$ for some Q' with $P' \approx Q'$, and
2. if $Q \stackrel{\alpha}{\longrightarrow} Q'$ then $P \stackrel{\alpha}{\Longrightarrow} P'$ for some P' with $P' \approx Q'$.

□

If P and Q are to be equal, each initial action of P must be matched by *at least one* action of Q, and vice versa. Subsequently, the agents need only be observation equivalent. Note that $P \sim Q$ implies $P = Q$, which in turn implies $P \approx Q$. Observation congruence establishes the equational theory of CCS [64], but we shall not expound that theory here.

B.2 Modal μ-calculus

The modal μ-calculus is a rich logic for expressing dynamic properties of systems. The logic was formulated by Pratt [81] and Kozen [52] as a generalisation of propositional dynamic logics (which themselves extend Hoare logics to recursive programs). Starting from a simple modal logic, put forward by Hennessy and Milner in [43] where $\Box$ and $\Diamond$ modalities are indexed (or relativised) by actions, Stirling shows how the modal μ-calculus naturally arises when one wishes to more abstractly express durable properties of communicating systems. Stirling, in [89], extends HML, and Kozen's μ-calculus in a minor, but very convenient manner: modalities are indexed by *sets* of actions. The resulting logic is very suitable for exploring temporal properties of parallel systems in general, including at least concurrent **while** programs, CSP and CCS programs, and Petri nets [91, 11].

Temporal logics have often been advanced as specification formalisms for concurrent programs—though they are suitable for purely sequential, non-interfering programs too. (See Pnueli [80], for example, for good motivation, and [28, 38] for recent surveys.) The problem then remains of verifying that models of programs satisfy their temporal logic specifications. Emerson and Lei [29] discovered a decidable method for a restricted version of the μ-calculus; Stirling and Walker [90] were first to describe a *local* model checker for the full logic. Local model checking is appealing since one may never need to construct the entire model to verify interesting properties of systems.

HENNESSY-MILNER LOGIC

Hennessy-Milner logic (HML) is a modal logic for specifying local capabilities of systems usually modelled in CCS. Typically, modal logics are interpreted over Kripke structures (unlabelled transition systems) but the modal connectives of HML are labelled by actions, so we interpret them over labelled transition systems. Since CCS terms define these structures, the relationship between logic and algebra is a natural one. Formulae of HML are constructed by the following syntax:

$$\Phi \quad ::= \quad tt \quad \Big| \quad \neg\Phi \quad \Big| \quad \Phi \wedge \Phi \quad \Big| \quad [K]\Phi$$

where $K \subseteq \mathcal{A}$. Other logical connectives are defined as required using negation. In particular $\langle K\rangle\Phi \stackrel{\text{def}}{=} \neg[K]\neg\Phi$ is the dual to the 'box' modality ('diamond'). This description somewhat generalises the original formulation of the logic [42, 43] since there only single actions are permitted in decorating modal connectives.

Now if $(\mathcal{P}, \mathcal{A}, \{ \stackrel{\alpha}{\longrightarrow} \mid \alpha \in \mathcal{A}\})$ is a labelled transition system we can precisely determine when some process $P \in \mathcal{P}$ enjoys a property expressed in HML by the following inductive definition of *satisfaction*. First, let $K(P)$ be the set of states reachable

from P via an action in K: $K(P) \stackrel{\text{def}}{=} \{P' \in \mathcal{P} \mid P \xrightarrow{a} P' \text{ for some } a \in K\}$. Then:

$$\begin{aligned}
&P \models tt \\
&P \models \neg\Phi \quad \text{iff} \quad P \not\models \Phi \\
&P \models \Phi \wedge \Psi \quad \text{iff} \quad P \models \Phi \text{ and } P \models \Psi \\
&P \models [K]\Phi \quad \text{iff} \quad \forall Q \in K(P).Q \models \Phi \\
&P \models \langle K\rangle\Phi \quad \text{iff} \quad \exists Q \in K(P).Q \models \Phi
\end{aligned}$$

Intuitively, any process satisfies the property *tt* (*true*); in contrast, no process satisfies *ff*. For the modal connectives, P satisfies $\langle K\rangle\Phi$ if and only if some K-derivative satisfies the property Φ; P satisfies $[K]\Phi$ if and only if all K-derivatives satisfy Φ.

Example B.4 The box modalities express necessities, the diamond modalities express capabilities. Consider the property $\langle a\rangle tt$ for the simple action a:

$$P \models \langle a\rangle tt \quad \text{iff} \quad \exists Q \in \{P' \in \mathcal{P} \mid P \xrightarrow{a} P'\}.Q \models tt$$

Since $Q \models tt$ for all a-derivatives of P (in particular, since it is true for all $P \in \mathcal{P}$) this property therefore expresses the capacity for P to perform an a action. In contrast $\neg\langle a\rangle tt = [a]ff$, and:

$$P \models [a]ff \quad \text{iff} \quad \forall P' \in \{P' \in \mathcal{P} \mid P \xrightarrow{a} P'\}.P' \models ff$$

But $Q \not\models ff$ for any Q. This property therefore expresses the fact that P cannot (immediately) perform an a action. In a similar vein, note that $\forall K,\ \mathbf{0} \models [K]ff$. ♣

There are a number of interesting results concerning HML, the most important of which is the Modal Characterisation Theorem. If the agents P and Q are finite state, they are strongly bisimilar if and only if they have the same modal properties: $P \sim Q \Leftrightarrow \{\Phi \mid P \models \Phi\} = \{\Psi \mid Q \models \Psi\}$. While HML is rich enough to express properties about *finite* behaviour, it is not expressive enough to capture enduring properties. The μ-calculus introduces a temporal operator to the modal logic for this purpose.

Modal Temporal Logic

An alternative way of assigning meaning to modal logic formulae is through *satisfaction sets*. We use the notation $\|\Phi\|^{\mathcal{P}}$ to denote the set of processes (in $\mathcal{P}$) having the property Φ. This set is defined inductively on the structure of formulae:

$$\begin{aligned}
\|tt\|^{\mathcal{P}} &= \mathcal{P} \\
\|\neg\Phi\|^{\mathcal{P}} &= \mathcal{P} - \|\Phi\|^{\mathcal{P}} \\
\|\Phi \wedge \Psi\|^{\mathcal{P}} &= \|\Phi\|^{\mathcal{P}} \cap \|\Psi\|^{\mathcal{P}} \\
\|[K]\Phi\|^{\mathcal{P}} &= \{P \in \mathcal{P} \mid K(P) \subseteq \|\Phi\|^{\mathcal{P}}\} \\
\|\langle K\rangle\Phi\|^{\mathcal{P}} &= \{P \in \mathcal{P} \mid \exists P' \in K(P) \cap \|\Phi\|^{\mathcal{P}}\}
\end{aligned}$$

As long as $\mathcal{P}$ is transition closed—*i.e.*, if $P \in \mathcal{P}$ then every derivative of P is in the set—then the two semantics of HML coincide: $P \models \Phi$ if and only if $P \in \|\Phi\|^{\mathcal{P}}$.

It is by extending these semantics that we assign meaning to formulae of the modal μ-calculus which introduces propositional variables and a fixed point operator to the modal logic:

$$\Phi \quad ::= \quad Z \quad \Big| \quad \neg\Phi \quad \Big| \quad \Phi \wedge \Phi \quad \Big| \quad [K]\Phi \quad \Big| \quad \nu Z.\Phi$$

A fixed point formula $\nu Z.\Phi$ represents the maximal solution to the (possibly) recursive modal equation $Z \stackrel{\text{def}}{=} \Phi$. This solution always exists, as does the minimal solution denoted by $\mu Z.\Phi \stackrel{\text{def}}{=} \neg\nu Z.\neg\Phi\{\neg Z/Z\}$, under the syntactic restriction that if Z appears free in the body Φ, it does so within the scope of an even number of negations. Models for this temporal logic are given as before by labelled transition systems—but now we have to supply an interpretation for the propositional variables appearing in subformulae. Let $\mathcal{V}$ be a *valuation* that assigns a subset of $\mathcal{P}$ to each variable Z (*i.e.*, the set of states having the property expressed by Z). Now:

$$\begin{aligned} \|Z\|^{\mathcal{P}}_{\mathcal{V}} &= \mathcal{V}(Z) \\ \|\nu Z.\Phi\|^{\mathcal{P}}_{\mathcal{V}} &= \bigcup\{\mathcal{E} \mid \mathcal{E} \subseteq \|\Phi\|^{\mathcal{P}}_{\mathcal{V}[\mathcal{E}/Z]}\} \\ \|\mu Z.\Phi\|^{\mathcal{P}}_{\mathcal{V}} &= \bigcap\{\mathcal{E} \mid \|\Phi\|^{\mathcal{P}}_{\mathcal{V}[\mathcal{E}/Z]} \subseteq \mathcal{E}\} \end{aligned}$$

with the meanings of the other logical connectives being given as above (but with respect to $\mathcal{V}$). The notation $\mathcal{V}[\mathcal{E}/Z]$ signifies the valuation $\mathcal{V}'$ that agrees with $\mathcal{V}$ on all variables except Z, for which $\mathcal{V}'(Z) = \mathcal{E}$.

Example B.5 Notice that we do not need to take *tt* or *ff* as primitive in the temporal logic: they can be *defined*. From the semantics:

$$\begin{aligned} \|\nu Z.Z\|^{\mathcal{P}}_{\mathcal{V}} &= \bigcup\{\mathcal{E} \mid \mathcal{E} \subseteq \|Z\|^{\mathcal{P}}_{\mathcal{V}[\mathcal{E}/Z]}\} \\ &= \bigcup\{\mathcal{E} \mid \mathcal{E} \subseteq \mathcal{E}\} \\ &= \mathcal{P} \end{aligned}$$

while, in contrast, $\|\mu Y.Z\|^{\mathcal{P}} = \emptyset$. But if a *property*—of the labelled transition system $(\mathcal{P}, \mathcal{A}, \{\stackrel{a}{\longrightarrow} \mid a \in \mathcal{A}\})$—is identified with a subset of $\mathcal{P}$, the only property $\mathcal{P}$ can express is the property *true*. Conversely, $\emptyset$ expresses *false*: no process can have this property. ♣

Notice that $\|\nu Z.Z\|^{\mathcal{P}}_{\mathcal{V}} = \mathcal{P}$ independently of any particular $\mathcal{V}$. The same is true of all HML formulae but, properly speaking, a temporal property is only expressed in a closed formula (no free Z) of the logic. We can thus continue to use the notation $\|\Phi\|^{\mathcal{P}}$ to represent the set of processes that have the property Φ. Now satisfaction is defined:

$P \models \Phi$ whenever $P \in \|\Phi\|^{\mathcal{P}}$. The notion may be generalised to sets of processes: $\mathcal{E} \models \Phi$ if and only if all $P \in \mathcal{E}$ satisfy Φ.

The modal μ-calculus is a very powerful logic, but suffers from cumbersome-looking syntax. A good reading of μ-calculus formulae only comes through tackling some examples! First, a difficult one:

Example B.6 The property $\mu Z.\Phi \vee \langle K \rangle Z$, when Z does not appear free in Φ, asserts that Φ *may* eventually hold. Now, dropping the (fixed) superscript $\mathcal{P}$, the semantics supply the interpretation:

$$\begin{aligned}
\|\mu Z.\Phi \vee \langle K \rangle Z\|_\nu &= \bigcap\{\mathcal{E} \mid \|\Phi \vee \langle K \rangle Z\|_{\nu[\mathcal{E}/Z]} \subseteq \mathcal{E}\} \\
&= \bigcap\{\mathcal{E} \mid \|\Phi\|_{\nu[\mathcal{E}/Z]} \cup \|\langle K \rangle Z\|_{\nu[\mathcal{E}/Z]} \subseteq \mathcal{E}\} \\
&= \bigcap\{\mathcal{E} \mid \|\Phi\|_{\nu[\mathcal{E}/Z]} \cup \{P \mid \exists P' \in K(P) \cap \mathcal{E}\} \subseteq \mathcal{E}\} \\
&= \bigcap\{\mathcal{E} \mid \|\Phi\|_{\nu[\mathcal{E}/Z]} \cup \{P \mid \exists P' \in \mathcal{E}.\exists a \in K.P \xrightarrow{a} P'\} \subseteq \mathcal{E}\}
\end{aligned}$$

So a process P satisfies the defining condition either if $P \models \Phi$ or some K-successor P' is in $\mathcal{E}$ *implies* $P \in \mathcal{E}$. So $P \models \mu Z.\Phi \vee \langle K \rangle Z$ if $P \models \Phi$ or, if it does not, some K-successor $P' \models \Phi$ or, if it does not, some K-successor P'' of P' satisfies $\Phi \ldots$: in short, some state reachable from P via zero or some finite number of K actions must satisfy Φ. ♣

The above is an example of a *liveness* property, asserting that something (good), characterised by the property Φ, may happen. Liveness properties are associated with least fixed points in the fixed point logic. On the other hand, *safety* properties reverse the scenario, asserting that something (bad) must never happen. These correspond to greatest fixed points.

Example B.7 The property $\neg\mu Z.\neg\Phi \vee \langle K \rangle Z = \nu Z.\Phi \wedge [K]Z$ is a strong invariance property: Φ holds along all computations involving only actions drawn from the set K.

$$\begin{aligned}
\|\nu Z.\Phi \wedge [K]Z\|_\nu &= \bigcup\{\mathcal{E} \mid \mathcal{E} \subseteq \|\Phi\|_{\nu[\mathcal{E}/Z]} \cap \|[K]Z\|_{\nu[\mathcal{E}/Z]}\} \\
&= \bigcup\{\mathcal{E} \mid \mathcal{E} \subseteq \|\Phi\|_{\nu[\mathcal{E}/Z]} \cap \{P \mid K(P) \subseteq \mathcal{E}\}\} \\
&= \bigcup\{\mathcal{E} \mid \mathcal{E} \subseteq \|\Phi\|_{\nu[\mathcal{E}/Z]} \cap \{P \mid P' \in K(P) \text{ implies } P' \in \mathcal{E}\}\}
\end{aligned}$$

This therefore specifies a set of states that is closed under K actions, but each of which satisfies the property expressed in Φ. ♣

Example B.8 Let $[a_1 \ldots a_n]\Phi \stackrel{\text{def}}{=} [\{a_1 \ldots a_n\}]\Phi$, $[-K]\Phi \stackrel{\text{def}}{=} [\mathcal{A} - K]\Phi$ and $[-]\Phi \stackrel{\text{def}}{=} [\mathcal{A}]\Phi$. We can specialise the examples above in interesting ways. Firstly, the fixed point formula $\nu Z.\Phi \wedge [-]Z$ expresses invariance of Φ over all computations—*i.e.*, the

branching time temporal logic formula **AG** Φ. Secondly, $\mu Z.\Phi \vee \langle\tau\rangle Z$ specifies that after some silent activity the property Φ may become true. This is usually written $\langle\!\langle\ \rangle\!\rangle\Phi$. Similarly $[\![\ ]\!]\Phi = \nu Z.\Phi \wedge [\tau]Z$. ♣

Notice the distinction between $\nu Z.\Phi \wedge [-]Z$ which is global invariance, and the case when the modality selects a subset of $\mathcal{A}$—local invariance. The latter property is also expressible in CTL, but only in a rather complicated and unintuitive formula.

The double bracket modalities express *weak* modal properties—they treat the silent τ action of CCS in much the same way as it is treated in the theory of observational equivalence. Generalising, we get:

$$\begin{aligned}
[\![K]\!]\Phi &\stackrel{\text{def}}{=} [\![\ ]\!][K][\![\ ]\!]\Phi \\
&= \nu Z.[K][\![\ ]\!]\Phi \wedge [\tau]Z \\
&= \nu Z.[K](\nu Y.\Phi \wedge [\tau]Y) \wedge [\tau]Z
\end{aligned}$$

and similarly $\langle\!\langle K\rangle\!\rangle\Phi = \neg[\![K]\!]\neg\Phi$:

$$\langle\!\langle K\rangle\!\rangle\Phi \stackrel{\text{def}}{=} \mu Z.\langle K\rangle(\mu Y.\Phi \vee \langle\tau\rangle Y) \vee \langle\tau\rangle Z$$

Just as Hennessy-Milner logic characterises (strong) bisimulation, Stirling has shown that the modal μ-calculus does likewise: if P and Q are (image finite) processes then they are strongly bisimilar if and only if they have the same temporal properties. Indeed, the *weak* μ-calculus, using only the double bracket modalities together with the fixed point and boolean connectives, characterises observation equivalence.

The examples considered above serve to illustrate the utility of macros! Yet although some of these formulae look complicated they barely scratched the surface of the expressive power of the logic [11]. The μ-calculus is a very powerful logic and in general the truth of assertions expressed in it is undecidable. However, there are polynomial time algorithms for deciding if an assertion is valid on a particular *finite* model.

Local Model Checking

The simple property checker described here is due to Stirling and Walker [90]. It is not very efficient as it stands, but Cleavland discovered significant improvements (see [19]) when he implemented local model checking in the Edinburgh Concurrency Workbench [20]. Local model checking contrasts with *global* model checking in that one need not *a priori* construct the model before checking that a particular state satisfies a formula of the logic. Of course, if one wants to prove a global invariant, then local model checking brings nothing new to the problem.

The temporal property checker is a *tableau* proof system, each rule used to construct the tree having the general form

$$\frac{\mathcal{E} \vdash \Phi}{\{\mathcal{E}_i \vdash \Phi_i\}}$$

where there may be side-conditions. The premise sequent, above the line, is the goal to be proved—*i.e.*, that the states $\mathcal{E} \subseteq \mathcal{P}$ satisfy Φ—and the subsequents are derived from this according to the form of Φ and the structure of the model in the vicinity of $\mathcal{E}$. The rules for the modal fragment of the logic, which we assume now is in positive form (that is, negation is omitted, and we take the derived operators $\vee$, $\langle K \rangle$, *etc.*, as primitive) are as follows:

$$\wedge\ \frac{\mathcal{E} \vdash \Phi \wedge \Psi}{\mathcal{E} \vdash \Phi \qquad \mathcal{E} \vdash \Psi} \qquad [K]\ \frac{\mathcal{E} \vdash [K]\Phi}{K(\mathcal{E}) \vdash \Phi}$$

$$\vee\ \frac{\mathcal{E} \vdash \Phi \vee \Psi}{\mathcal{E}_1 \vdash \Phi \qquad \mathcal{E}_2 \vdash \Psi} \qquad \langle K \rangle\ \frac{\mathcal{E} \vdash \langle K \rangle \Phi}{f(\mathcal{E}) \vdash \Phi}$$

with side conditions on the $\vee$-rule and the $\langle K \rangle$-rule which introduce choices. Note first that $K(\mathcal{E})$ generalises the earlier notation: $K(\mathcal{E}) = \{P' \in \mathcal{P} \mid \exists P \in \mathcal{E}.\exists a \in K.P \xrightarrow{a} P'\}$. The side condition in the $\vee$-rule is just that $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2$. In the case that $\mathcal{E}$ is a single state the choice here is over which branch of the disjunct to take: this more general formulation is more powerful, but not readily automated. The choice in the $\langle K \rangle$-rule is similar: $f : \mathcal{E} \to K(\mathcal{E})$ allows one to discard as many successors from $\mathcal{E}$ as desired.

The essential ingredient in the rules for the fixed point formulae is the use of *propositional constants*, these being introduced as fixed point operators are encountered in traversing the structure of the formula. Let U range over such constants, and let $\sigma Z.$ stand or either fixed point operator. The final rules are:

$$\sigma Z.\ \frac{\mathcal{E} \vdash \sigma Z.\Phi}{\mathcal{E} \vdash U} \qquad \text{introducing } U \stackrel{\text{def}}{=} \sigma Z.\Phi \text{ and } U \text{ is a fresh variable;}$$

$$U\ \frac{\mathcal{E} \vdash U}{\mathcal{E} \vdash \Phi\{U/Z\}} \qquad \text{as long as } U \stackrel{\text{def}}{=} \sigma Z.\Phi.$$

When a fixed point formula is encountered it is replaced by a (new) constant, and when a constant is subsequently encountered it is replaced by the body of its definition with U replacing all free occurrences of Z in the body.

To see if each of a set of processes $\mathcal{E}$ has the property Φ, the model checker is invoked with the root sequent $\mathcal{E} \vdash \Phi$. This process would not terminate because of the constant rule, but there are several conditions to check whether a node $\mathcal{F} \vdash \Psi$ should be considered a terminal (a leaf, in the tree constructed):

1. the set of states is trivial $\mathcal{F} = \emptyset$;
2. $\Psi = \langle K \rangle \Phi$ and for some $P \in \mathcal{F}$, $K(P) = \emptyset$;
3. $\Psi = U \stackrel{\text{def}}{=} \nu Z.\Phi$ and some sequent $\mathcal{E} \vdash U$, with $\mathcal{F} \subseteq \mathcal{E}$, appears higher in the proof tree;
4. $\Psi = U \stackrel{\text{def}}{=} \mu Z.\Phi$ and some sequent $\mathcal{E} \vdash U$, with $\mathcal{E} \subseteq \mathcal{F}$, appears higher in the proof tree.

No rule applies to a terminal. A node fulfilling the first and third of these conditions is deemed to be *successful* terminal, not otherwise. A successful tableau is a finite proof tree that has only successful terminals. The crucial results (see [90] for the details) are as follows:

Proposition B.1 If $\mathcal{E} \vdash \Phi$ has a successful tableau then $\mathcal{E} \models \Phi$. □

Proposition B.2 If $\mathcal{E}$ is a finite set of finite state processes and $\mathcal{E} \models \Phi$, then $\mathcal{E} \vdash \Phi$ has a successful tableau. □

The first of these demonstrates that the proof system is sound, the latter that is complete if the models are finite (when it is also decidable). Stirling and Bradfield [10, 11] extend the proof system described above to arbitrary labelled transition systems: their method remains sound and complete, but not decidable in general (of course).

Example B.9 To illustrate the working of the model checker, consider the very trivial CCS model $\mathit{Fix}\,(X.\, a.X + b.\mathbf{0})$ which has just the two states $\{X, \mathbf{0}\}$. The property $\nu Z.\langle b \rangle \mathit{tt} \wedge [-b]Z$ obviously holds at X. The tableau:

$$\dfrac{\dfrac{\dfrac{\dfrac{\{X\} \vdash \nu Z.\langle b \rangle \mathit{tt} \wedge [-b]Z}{\{X\} \vdash U}}{\{X\} \vdash \langle b \rangle \mathit{tt} \wedge [-b]U}}{\dfrac{\{X\} \vdash \langle b \rangle \mathit{tt}}{\{X\} \vdash \mathit{tt}} \qquad \dfrac{\{X\} \vdash [-b]U}{\{X\} \vdash U}}}$$

demonstrates this. Note that $\{X\} \vdash \langle b \rangle \mathit{tt}$ is not terminal, and we disregard this transition in moving to the sequent below. On the other branch, $\{X\} \vdash [-b]U$ has only one $(-b)$-successor to be considered.

Strictly speaking the left-hand leaf is not terminal, but this subtree does terminate successfully. Recall that $\mathit{tt} \stackrel{\text{def}}{=} \nu Z.Z$, so by the fixed point and constant rules we arrive at a sequent similar to that terminating the right-hand branch.

The right-hand terminal is successful because $\{X\} \vdash U$ appears higher up in the proof tree. $\{X\} \models \nu Z.\langle b \rangle \mathit{tt} \wedge [-b]Z$ follows from Proposition B.1. ♣

In the example it was not necessary to construct all states of the model in order to prove the result. The same would not be true of the global invariant $\nu Z.\langle - \rangle tt \wedge [-]Z$ which expresses freedom form deadlock. In that case, all proof trees contain the unsuccessful terminal $\{X, \mathbf{0}\} \vdash \langle - \rangle tt$.

B.3 HOL Proofs

HOL TACTICS FOR GEOGRAPHIC DATA

```
fun HANDY_TAC thl =
         ASM_REWRITE_TAC thl
    THEN POP_ASSUM_LIST (fn _ => ALL_TAC)
    THEN REPEAT STRIP_TAC
    THEN RES_TAC
    THEN ASM_REWRITE_TAC[]
    THEN NO_TAC;
```

Commentary on `HANDY_TAC` Given a list of rewrite theorems, the goal expected is $[H] \vdash F_i \Rightarrow F_i'$, where F_i is a simple term derived from **F**. First rewrite with the given assumptions and rewrite theorems, then discard the assumptions to render something like $? \vdash (f \Rightarrow g) \Rightarrow f' \Rightarrow g'$. Strip all leading quantifiers (*i.e.*, by applying Elimination rules) from the goal, and resolve among these new assumptions. Finally, rewrite (g') using these to solve the goal completely—otherwise this tactic fails (to be handy).

```
fun PIMP th = REWRITE_TAC[MX2_TH0,MX2_TH1] THEN          (*1*)
  (let val tm = concl th
    in if is_comb tm                                     (*2*)
       then
         let val n =
             (#Name o dest_const o #Rator o dest_comb) tm
          in case n of
               "PT"  => MATCH_ACCEPT_TAC PT_L1 ORELSE   (*2a*)
                        MATCH_ACCEPT_TAC PT_L2 ORELSE
                        HANDY_TAC[PT,PC]                (*2b*)
             | "MX4" => MATCH_ACCEPT_TAC MX4_L1 ORELSE  (*2c*)
                        MATCH_ACCEPT_TAC MX4_L2 ORELSE
                        MATCH_ACCEPT_TAC MX4_L3 ORELSE
                        MATCH_ACCEPT_TAC MX4_L4
             | _     => HANDY_TAC[RT1,RC]               (*2d*)
         end
       else ALL_TAC                                      (*3*)
   end) handle _ => HANDY_TAC[];                         (*4*)
```

Commentary on `PIMP` The goal `PIMP` expects is $[H] \vdash F_i \Rightarrow F_i'$, where F_i is a simple term derived from **F**, so:

1. If F_i is derived from MX2, facts $\vdash \mathsf{MX2}(a, \mathsf{T})$ and $\vdash \mathsf{MX2}(\mathsf{T}, b)$ should solve it.

2. Otherwise, given $[H] \vdash F(a) \Rightarrow F(a')$ find out what F is:

(a) a PT so match the goal with $\vdash \mathsf{PT}(p,[a;b]) \Rightarrow \mathsf{PT}(p,[\mathsf{T};b])$ *etc.*;

(b) still a PT, but let `HANDY_TAC` try to solve this more general case.

(c) F is an MX4 so match $\vdash \mathsf{MX4}[a,b,c,d] \Rightarrow \mathsf{MX4}[\mathsf{T},b,c,d]$, *etc.*.

(d) Otherwise it is an additional **RT** term (*i.e.*, RT1) which `HANDY_TAC` can solve.

3. `ALL_TAC` is like `skip` (the program never reaches this when $\mathbf{F} = \mathbf{MX} \wedge \mathbf{PT} \wedge \mathbf{RT}$).

4. Trap exceptions from attempts to decompose $(b \Rightarrow a) \Rightarrow b' \Rightarrow a'$ which are from a regular RT terms. Another job for `HANDY_TAC`.

```
val RSTRIP_TAC = STRIP_TAC THEN
    POP_ASSUM_LIST
       (fn thl => MAP_EVERY (fn th => ASSUME_TAC th) thl);
```

Commentary on `RSTRIP_TAC` This strips the goal, an implication where used, and reverses the order of the assumptions (conjunctions in the antecedent of the goal) so obtained.

```
local
  fun POP_ONE th =
      ASSUME_TAC th THEN UNDISCH_TAC(concl th) THEN PIMP th

  fun RECURSE () = POP_ASSUM (fn th => STRIP_TAC
      THENL
        [POP_ONE th,
         RECURSE () ORELSE POP_ASSUM (fn th => POP_ONE th)])
in
  val SRR_TAC =
           VC_TAC THENL[ALL_TAC,MATCH_ACCEPT_TAC FB_IMP_F]
      THEN RSTRIP_TAC
      THEN RECURSE ()
end;
```

Commentary on `SRR_TAC` See Section 5.4.

```
fun repeat 0 tac = ALL_TAC
  | repeat n tac = tac THEN (repeat (n-1) tac);

fun PRR_TAC F pfm_list n = VC_TAC F
    THEN TRY (MATCH_ACCEPT_TAC FB_IMP_F)
    THEN REWRITE_TAC (PT_DEFS @ RT_DEFS @ pfm_list)
    THEN STRIP_GOAL_THEN (fn th =>
                             REWRITE_TAC(CONJUNCTS th)
                        THEN ASSUME_TAC th
                        THEN UNDISCH_TAC(concl th))
    THEN STRIP_TAC
    THEN (repeat n) RES_TAC
    THEN ASM_REWRITE_TAC (PT_THMS @ MX_THMS)
end;
```

Commentary on `PRR_TAC` See Section 5.4. This version is more general, passing a copy (the `F` parameter) of the invariant on to `SEQ_TAC` which implements heuristics to guess the strongest predicate to assert between sequenced commands. The `pfm_list` is an (optional) list of rewrite rules used to simplify the disjunctive points "free to move" conditions (to be used with great care since these are free axioms). The idea is to first prove that if *the points P_i are controlled normal (respectively, reverse)* then *they are free to move normal (reverse)* is invariant, and then to use the fact $p \Rightarrow q \Rightarrow (p \vee q = q)$ to obtain the simplifying rewrite rule(s) for the proof that **F** is invariant. This is quicker than proving $\mathbf{P} \wedge \mathbf{F}$ and constructing `pfm_list` on-the-fly, but less secure since the rewrite rules have to be entered as free axioms—*i.e.*, they cannot be deduced from $\vdash \{\mathbf{P}\}\, c\, \{\mathbf{P}\}$.

Appendix C

Examples of Geographic Data

This final appendix lists Geographic Data for some of the interlockings studied in the main text of the thesis. Pages 211–212 list the *PFM*, *PRR*, and sub-route release data for WEST (Figure C.1). Pages 214–215 list the same data files for the EAST-WEST interlocking in Figure C.2. These data extend that for WEST in the obvious way for a *single* SSI—the data for the protocol studied in Chapter 6 have not been listed here. Figures C.3 and C.4 display two further interlockings studied: the former was the subject of the Alvey Forest project [4, 8]; the latter is a fanciful scheme based loosely on Thornton Junction, in Fife, Scotland, and the author acknowledges the inspiration of [49] as the source for this scheme.

Finally, Figure C.5 is an artistic impression of the Leamington Spa scheme, as gleaned from the Geographic Data. The data in Figure 7.3 refer to marked routes `R35(2M)` which is the main route from signal `S35` to signal `S37` with a choice of overlaps through `P224` and `P225`. The other route, `R41(3M)` is the main route from signal `S41` to signal `S57`.

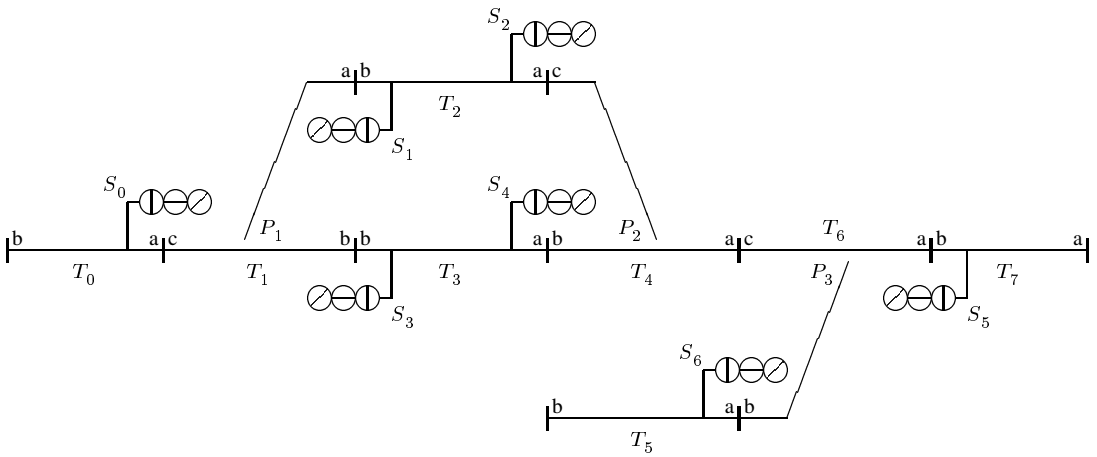


Figure C.1: Sample interlocking: WEST

```
/ Points Free to Move Data for Sample Interlocking: West

P1R    T1cb f , T1bc f , T1 c \

P1N    T1ca f , T1ac f , T1 c \

P2R    T4ba f , T4ab f , T2 c \

P2N    T4ca f , T4ac f , T2 c \

P3R    T6ac f , T6ca f , T3 c \

P3N    T6ab f , T6ba f , T3 c \
```

```
/ Panel Route Requests Data for Sample Interlocking: West

*Q02    if   P1 crf , T1ac f , T2ab f
        then P1 cr  , R02 s  , T1ca l , T2ba l \ .

*Q04    if   P1 cnf , T1bc f , T3ab f
        then P1 cn  , R04 s  , T1cb l , T3ba l \ .

*Q1     if   P1 crf , T0ba f , T1ca f
        then P1 cr  , R1 s   , T1ac l , T0ab l \ .

*Q2     if   P2 crf , P3 cnf , T7ab f , T4ac f
        then P2 cr  , P3 cn  ,
             R2 s   , T4ca l , T6ca l , T7ba l \ .

*Q3     if   P1 cnf , T0ba f , T1cb f
        then P1 cn  , R3 s   , T1bc l , T0ab l \ .

*Q4     if   P2 cnf , P3 cnf , T7ab f , T4ab f
        then P2 cn  , P3 cn  ,
             R4 s   , T4ba l , T6ca l , T7ba l \ .

*Q51    if   P2 crf , P3 cnf , T6ca f , T2ba f
        then P2 cr  , P3 cn  ,
             R51 s  , T6ac l , T4ac l , T2ab l \ .

*Q53    if   P2 cnf , P3 cnf , T6ca f , T3ba f
        then P2 cn  , P3 cn  ,
             R53 s  , T6ac l , T4ab l , T3ab l \ .

*Q5     if   P2 crf , P3 crf , T6ba f , T5ba f
        then P2 cr  , P3 cr  , R5 s   , T6ab l , T5ab l \ .

*Q6     if   P3 crf , T6ab f , T7ab f
        then P3 cr  , R6 s   , T6ba l , T7ba l \ .
```

```
/ Subroute Release Data for Sample Interlocking: West

T0ab f   if T0 c , T1ac f , T1bc f \ .

T5ab f   if T5 c , T6ab f \ .

T3ab f   if T3 c , T4ab f \ .
T3ba f   if T3 c , T1cb f \ .

T2ab f   if T2 c , T4ac f \ .
T2ba f   if T2 c , T1ca f \ .

T1ac f   if T1 c , R1  xs \ .
T1ca f   if T1 c , R02 xs \ .

T1cb f   if T1 c , R04 xs \ .
T1bc f   if T1 c , R3  xs \ .

T4ab f   if T4 c , T6ac f \ .
T4ba f   if T4 c , R4  xs \ .

T4ac f   if T4 c , T6ac f \ .
T4ca f   if T4 c , R2  xs \ .

T6ab f   if T6 c , R5  xs \ .
T6ba f   if T6 c , R6  xs \ .

T6ac f   if T6 c , R51 xs , R53 xs \ .
T6ca f   if T6 c , T4ba f , T4ca f \ .

T7ba f   if T7 c , T6ba f , T6ca f \ .
```

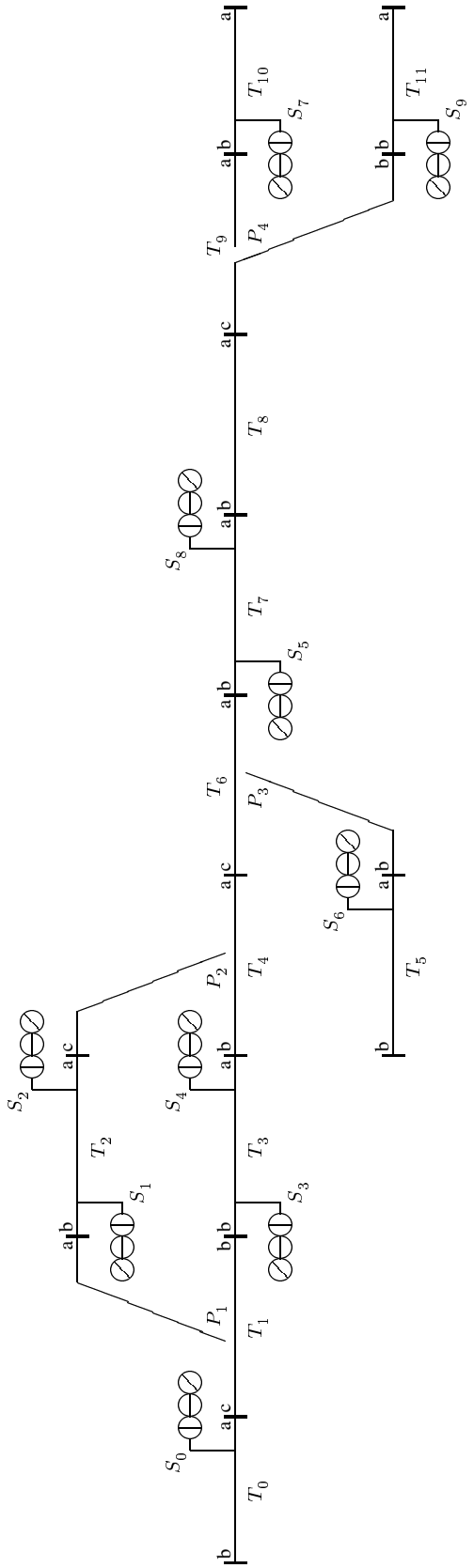


Figure C.2: The EASTWEST interlocking

```
/ Points Free to Move Data for Sample Interlocking: EastWest

/ As for West, but including:

P4R    T9cb f , T9bc f , T4 c \

P4N    T9ca f , T9ac f , T4 c \
```

```
/ Panel Route Requests Data for Sample Interlocking: EastWest

*Q02    if   P1 crf , T1ac f , T2ab f
        then P1 cr  , R02 s  , T1ca l , T2ba l \ .

*Q04    if   P1 cnf , T1bc f , T3ab f
        then P1 cn  , R04 s  , T1cb l , T3ba l \ .

*Q1     if   P1 crf , T0ba f , T1ca f
        then P1 cr  , R1 s   , T1ac l , T0ab l \ .

*Q28    if   P2 crf , P3 cnf , T7ab f , T4ac f
        then P2 cr  , P3 cn  ,
             R28 s  , T4ca l , T6ca l , T7ba l \ .

*Q3     if   P1 cnf , T0ba f , T1cb f
        then P1 cn  , R3 s   , T1bc l , T0ab l \ .

*Q48    if   P2 cnf , P3 cnf , T7ab f , T4ab f
        then P2 cn  , P3 cn  ,
             R48 s  , T4ba l , T6ca l , T7ba l \ .

*Q51    if   P2 crf , P3 cnf , T6ca f , T2ba f
        then P2 cr  , P3 cn  ,
             R51 s  , T6ac l , T4ac l , T2ab l \ .

*Q53    if   P2 cnf , P3 cnf , T6ca f , T3ba f
        then P2 cn  , P3 cn  ,
             R53 s  , T6ac l , T4ab l , T3ab l \ .

*Q5     if   P2 crf , P3 crf , T6ba f , T5ba f
        then P2 cr  , P3 cr  , R5 s   , T6ab l , T5ab l \ .

*Q68    if   P3 crf , T6ab f , T7ab f
        then P3 cr  , R68 s  , T6ba l , T7ba l \ .

*Q75    if   P4 crf , T9ca f , T7ba f
        then P4 cr  , R75  s , T9ac l , T8ab l , T7ab l \ .

*Q8a    if   P4 crf , T8ab f , T10ab f
        then P4 cr  , R8a s  , T8ba l , T9ca l , T10ba l \ .

*Q8b    if   P4 cnf , T8ab f , T11ab f
        then P4 cn  , R8b s  , T8ba l , T9cb l , T11ba l \ .

*Q95    if   P4 cnf , T9cb f , T7ba f
        then P4 cn  , R95 s  , T9bc l , T8ab l , T7ab l \ .
```

```
/ Subroute Release Data for Sample Interlocking: EastWest

/ As for West, with the following changes and additions:

T4ba f   if T4 c , R48 xs \ .

T4ca f   if T4 c , R28 xs \ .

T6ba f   if T6 c , R68 xs \ .

T7ba f   if T7 c , T6ba f , T6ca f \ .

T8ab f   if T8 c , T9ac f , T9bc f \ .
T8ba f   if T8 c , R8a xs , R8b xs \ .

T9ac f   if T9 c , R75 xs \ .
T9ca f   if T9 c , T8ba f \ .

T9cb f   if T9 c , T8ba f \ .
T9bc f   if T9 c , R95 xs \ .

T10ba f  if T10 c , T9ca f \ .
T11ba f  if T11 c , T9cb f \ .
```

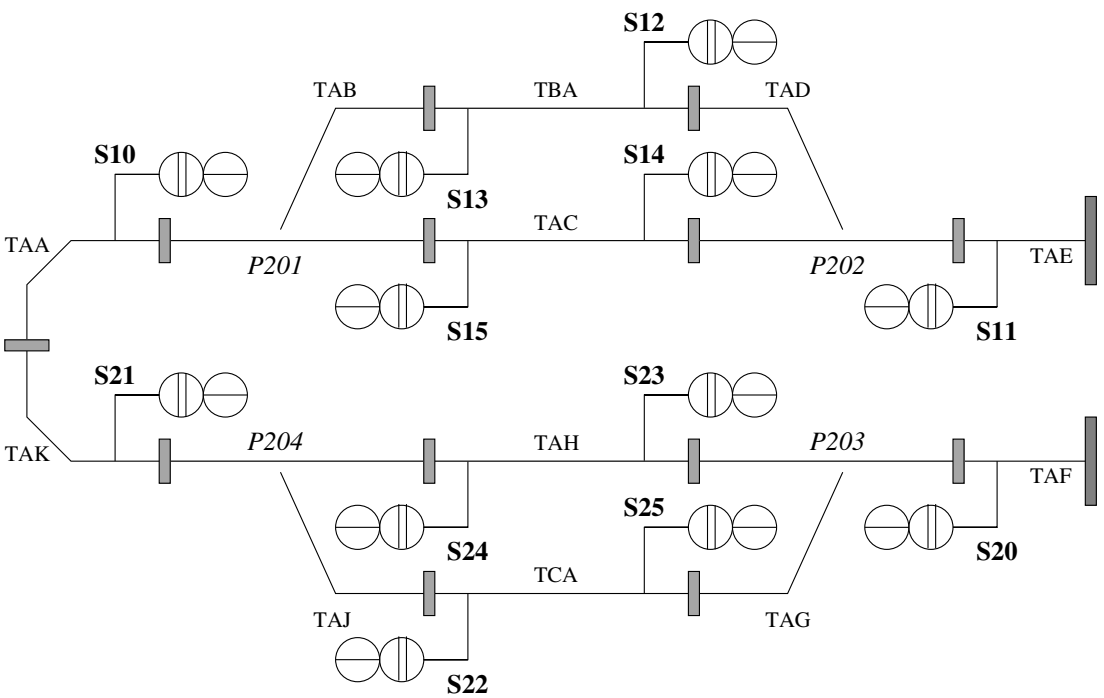


Figure C.3: The FOREST LOOP interlocking

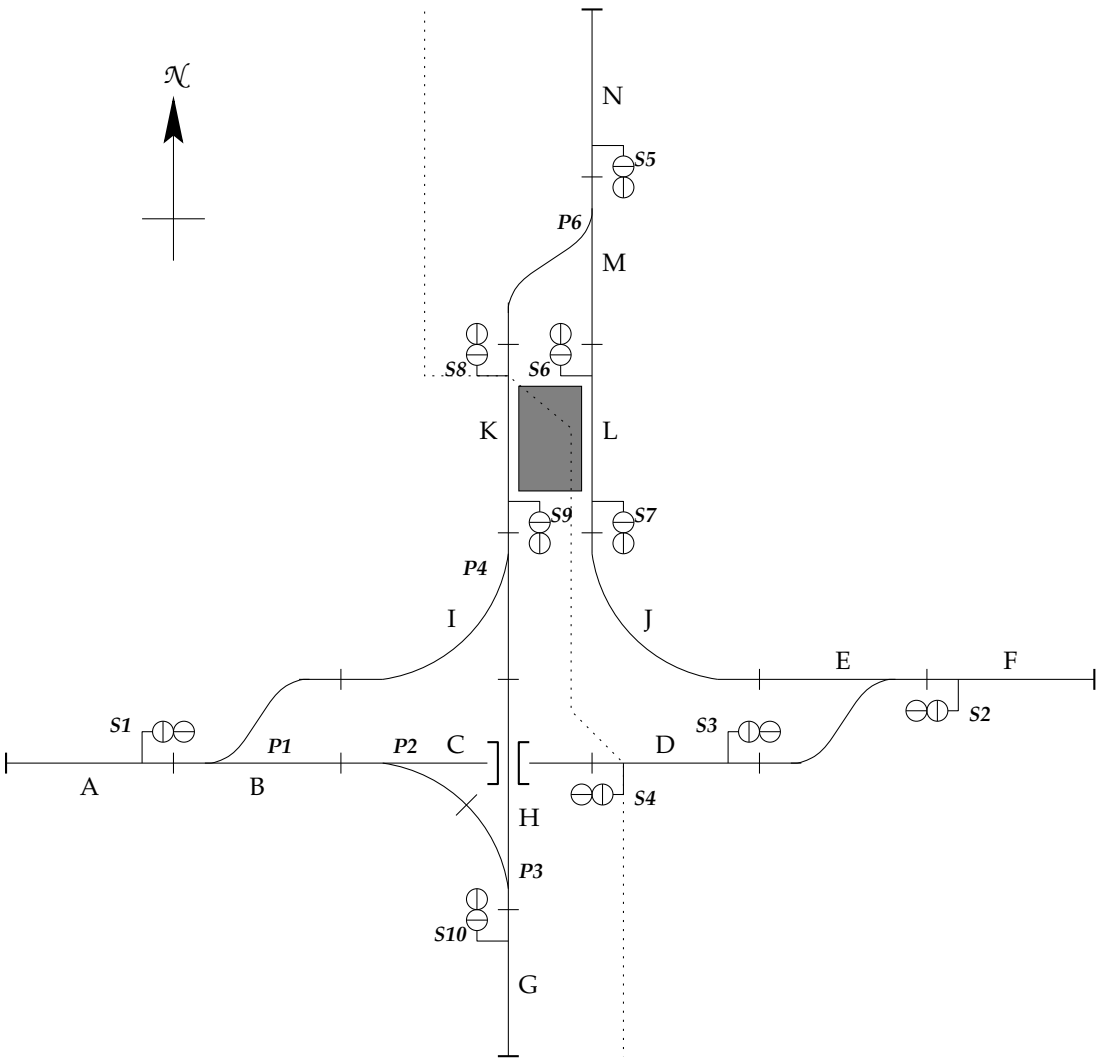


Figure C.4: The THORNTON JN. interlocking

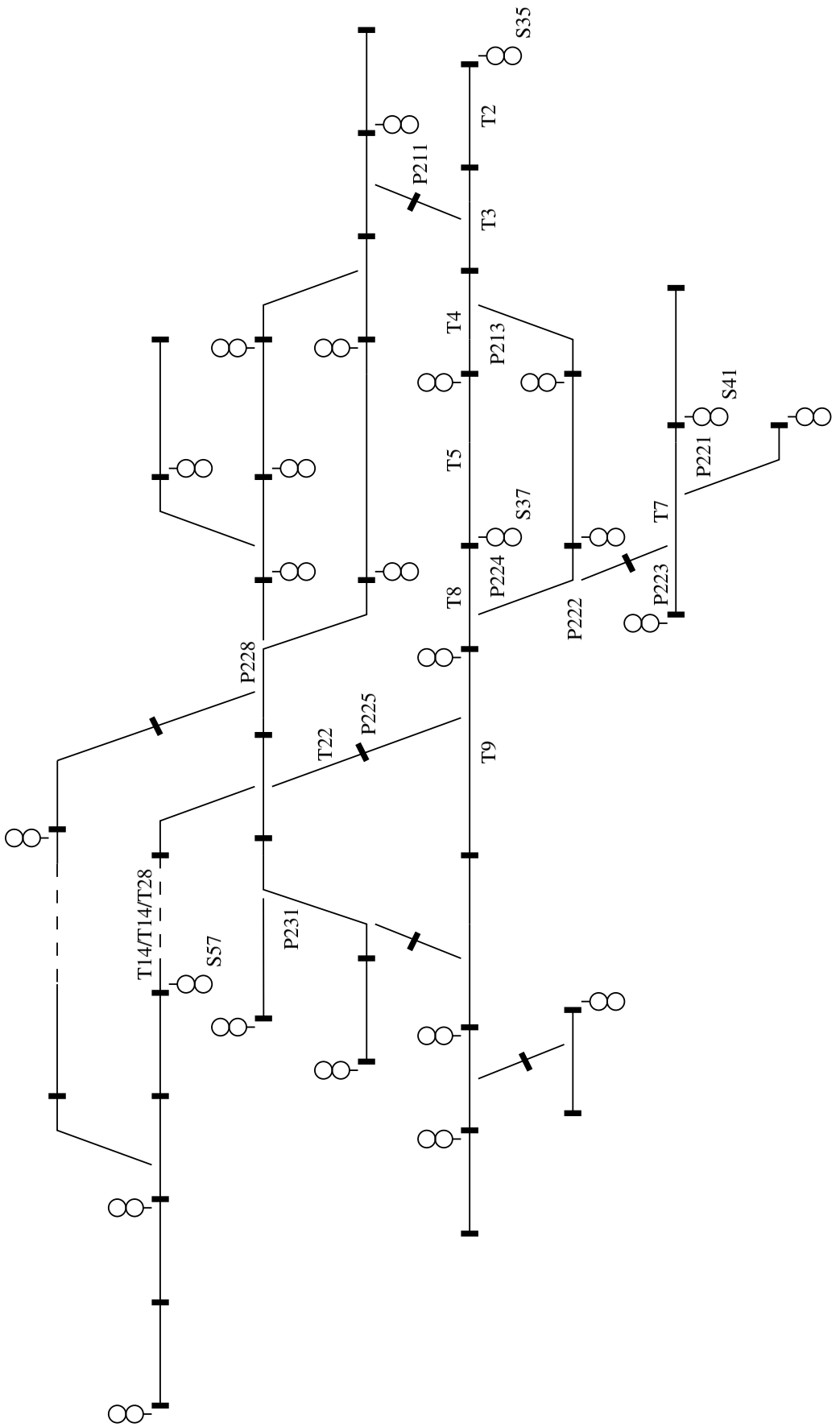


Figure C.5: An artistic impression of Leamington Spa