FEATURE INTEGRATION AND ITS LOGICAL PROPERTIES

by

HANNAH HARRIS

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

School of Computer Science
Faculty of Science
University of Birmingham
May 2005
Abstract

In this thesis, we have formalised a relationship between a well-known software problem called the feature interaction problem and a branch of mathematical logic known as theory change. In particular, we establish a direct correlation between the operations of feature integration and update, by drawing upon their inherent nonmonotonicity.

We achieve this by taking SMV Feature Integrator (SFI), a tool which automates feature integration on systems described using the model checker SMV, and formulating it in propositional logic. This enables us to give evidence of a correlation between the two operations by proving that the eight rationality postulates for the update operation hold in the context of SMV feature integration.

We then go on to construct a new update operator upon two arbitrary propositional logic formulae, $\psi$ and $\mu$, which specialises to SMV feature integration when $\psi$ and $\mu$ represent an SMV program and feature respectively.

This result means that in the future we could begin to make use of a representation theorem associated with update which allows us to reason about orderings on models.Potentially, this could enable us to consider features in terms of their ‘closeness’ to the base system.

This work provides both an interesting new application area for the logic of theory change, and a theoretical underpinning for the feature interaction problem which has a largely practical basis.
Acknowledgements

First and foremost, I would like to thank Mark Ryan, my supervisor. He has provided me with unstinting support, both intellectual and emotional, since I embarked on this journey and I could certainly not have achieved this without him.

I am also very grateful to the members of my thesis group, especially Manfred Kerber, and to Peter Hancox, for their encouragement and constructive comments.

I must also mention all the colleagues and friends who helped along the way, either by listening to unintelligible ranting about theorem-proving, taking me to the Mud Café for lunch or chatting in the coffee room. In particular, Stefano Cattani, Jon Rowe, Jeremy Wyatt, Kirsty Crombie-Smith, Nikos Gorogiannis, Aidan Harding, Dave Parker, Gavin Brown, Adrian Hartley and probably many others.

And from further afield, I must thank Pierre-Yves Schobbens, Alice Miller, Muffy Calder, Ken Turner and Stephan Reiff-Marganiec for their comments and support.

Finally, I am indebted to my parents for their constant love, patience and understanding, to Gabriel for putting up with me, and to Jonathan for supporting this undertaking in the first place.

This work is dedicated to my little girl Matilda, who has just about been in existence since all this began and who makes me smile, no matter what. I thank her for giving me the strength to see this through.
Contents

1 Introduction .................................................. 5

2 The Feature Interaction Problem ......................... 8
   2.1 Background .............................................. 8
   2.2 Feature Integration ..................................... 10
      2.2.1 Existing Methods of Feature Integration .......... 11
      2.2.2 Issues of Nonmonotonicity .......................... 13
   2.3 Approaches to the Problem .............................. 14
      2.3.1 Existing Methods of Feature Interaction Detection .... 15
      2.3.2 A Summary of Approaches ........................... 16
   2.4 SMV and the Feature Construct ....................... 19
      2.4.1 Introduction to SMV ............................... 19
      2.4.2 SFI: The SMV Feature Integrator ................. 21

3 The Logic of Theory Change ................................ 25
   3.1 Belief Revision .......................................... 26
      3.1.1 Preliminaries ...................................... 27
      3.1.2 The AGM Postulates for Revision .................. 28
      3.1.3 The Preferential Models Approach ................. 30
   3.2 Update .................................................. 32
      3.2.1 Katsuno and Mendelzon’s Rationality Postulates for Update ... 33
      3.2.2 Representation Theorem ............................ 34
      3.2.3 The Difference between Update and Revision ....... 35
3.3 Other Operations of Theory Change ........................................ 37
   3.3.1 Screened Revision ..................................................... 37
   3.3.2 Local Change .......................................................... 41
   3.3.3 Iterated Belief Revision .............................................. 42
3.4 Nonmonotonic Reasoning .................................................... 42
   3.4.1 Preliminaries ........................................................... 42
   3.4.2 Logical Properties .................................................... 43
   3.4.3 Nonmonotonic Reasoning and Belief Revision ............... 46

4 Propositional Logic Denotation of SMV ............................... 48
4.1 An Abstract Representation .............................................. 49
4.2 A Natural Denotation ...................................................... 54
4.3 The Correct Denotation ................................................... 57
4.4 Evidence of a Correlation ............................................... 61
4.5 An Equivalent Denotation ................................................. 69
4.6 What do the Update Axioms mean for SMV Feature Integration? ... 73

5 A ◦ Operator for SMV Feature Integration ............................. 76
5.1 Preliminaries ............................................................... 77
5.2 Definition of the ◦ Operator .............................................. 80
5.3 Proof of Correct Behaviour .............................................. 81
5.4 Proof of Update Axioms .................................................. 85
5.5 Minimality in Terms of Programs and Features. .................. 92

6 Conclusions and Directions for Future Work ....................... 95

A Hand-Coded Examples of Update Using the Results in this Thesis 98

B A Photograph ............................................................... 106
Chapter 1

Introduction

The aim of this thesis is to prove that a formal relationship exists between a well-known software problem called the feature interaction problem and a branch of mathematical logic known as theory change. In particular, we establish a direct correlation between the operations of feature integration and update.

A feature of a reactive system is a unit of functionality which may be added to the system in order to extend or modify its behaviour. Feature integration occurs when a system is modified in this way. For example, features of a telephone system might be Call Waiting, Call Screening and Ring Back When Free. Other examples may be found in lift systems, email and IP telephony.

The feature interaction problem is exhibited by a system when two or more features interact in an unexpected or undesirable way. It is ubiquitous in telecommunications and software systems and instances of its manifestation can be found in many different areas. Similarly, over the past decade, numerous varied approaches to the problem have been developed. Most methods focus on integration of features and detection of interactions between them. Other approaches are concerned with feature interaction resolution, and yet others with the prevention of feature interactions altogether. However, recall that the research presented here homes in upon the operation of feature integration. Although it is hoped that, in focussing upon this operation, we might cast the feature interaction problem in a new light, this is left to future research.

Update, then, is an operation of theory change which is performed upon a propositional
logic knowledge base (KB) when a new piece of information is to be added to it. In general, if the new information is implied by the existing KB, then the KB can simply be expanded by the new information. Otherwise, the eight rationality postulates for update specify exactly how the KB should be modified in order to incorporate the new information whilst maintaining consistency.

In examining the relationship between feature integration and update, we are drawing upon the inherently nonmonotonic nature of both operations. In short, an operation is nonmonotonic if it has the potential to disrupt properties of the system upon which it is performed. So feature integration is nonmonotonic because, on integration, a feature might disrupt, or override, properties of the system which were true previously. Update is nonmonotonic because in order to make a knowledge base consistent, certain facts may have to be retracted in order to update successfully by a new piece of information. In part, it was the potential to exploit the operations' nonmonotonicity which indicated that a formal link between them might manifest itself.

Informally, a link between the two areas is simple to demonstrate. Theory change involves revision of a knowledge base by new information; if the new information is consistent with the knowledge base, it can simply be added, but if it is not consistent, the knowledge base must be modified in order to resolve the inconsistency. Feature integration is extension or modification of a system in order to introduce new behaviour; usually the new system properties will cause some existing properties to be revised in some way.

The motivation behind this research is threefold. Firstly, this topic is compelling because of the intuitive link between two apparently disparate fields. Secondly, we have found an interesting new application area for nonmonotonic reasoning and theory change. Finally, we are attempting to give a theoretical underpinning to an area which has, up until now, had a largely practical basis.

There are several factors which render this research non-trivial, the principal ones being that: (1) the logic of features within reactive systems naturally tends to be temporal, whereas theory change is really only developed within the confines of propositional logic; (2) in order to make use of the semantic machinery which comes with the update operation, it is necessary to identify exactly what happens during feature integration in the context
of propositional logic.

In this thesis, we will explain how we have tackled the issues outlined above in order to prove that there is a formal link between the two areas in question. We have taken a feature construct for the model checker SMV, which has been successful in its application to both a telephone system and a lift system. We have then defined a translation procedure for SMV programs and features into propositional logic, and proved that the eight rationality postulates for update hold in this context. This gives evidence of a correlation between the two areas, and we further generalise this result by constructing an update operator which exactly defines SMV feature integration, thereby making available the semantic machinery associated with update.

In Chapters 2 and 3, we introduce the feature interaction problem and the logic of theory change respectively, as well as related work in each area. Chapters 4 and 5 constitute the main contribution of this thesis. In Chapter 4, we present our propositional logic denotation of SMV and the feature construct, and show evidence of a correlation between the two areas. Then, in Chapter 5, we generalise this result by constructing the new update operator and briefly discuss programs and features in terms of a representation theorem associated with update. The work presented in Appendix A serves to support the results of Chapters 4 and 5 by showing hand-coded examples of feature integration and update using the translation of SMV into propositional logic. It is not essential to read the appendix as it is simply an explanatory worked example. Finally, we give our conclusions, and indicate possible directions for future work in Chapter 6.

**Contribution of the Author**

The principal ideas presented in this thesis were initially formulated at meetings between the author, Hannah Harris, and her supervisor, Dr. Mark Ryan. The details of the work, the vast majority of the proofs, and the conference papers, [HR02] and [HR03], were all written by the author herself.
Chapter 2

The Feature Interaction Problem

In this chapter, we introduce the feature interaction problem and we review the area as a whole, paying particular attention to the subtopics involved in this research project. In Section 2.1, we discuss the feature interaction problem in general terms and give several examples in reactive systems. Then, in Section 2.2, we focus on the operation of feature integration and explain its importance in relation to this research. We give an overview of past approaches to the feature interaction problem in Section 2.3 and then present SMV and its associated feature construct in Section 2.4.

2.1 Background

A feature is a unit of functionality which may be integrated into a base system in order to extend or modify the system’s behaviour. Feature interaction refers to emergent system behaviour which arises as a result of the integration of two or more features. This emergent behaviour is likely to be unexpected in the sense that it does not appear in the specifications of either the base system or any of the features. Therefore, in the majority of cases, interactions are undesirable. Once interactions have been detected, methods of resolving them are implemented. A classic example is of a telephone system.

Example 2.1 Consider POTS (a Plain Old Telephone System) and let three of its subscribers be called A, B and C. Two possible features of POTS are CFB and CW. CFB
(Call Forward when Busy) is a feature whereby A forwards all calls to some subscriber B when engaged in another call. CW (Call Waiting) is invoked for B when A calls B and B is engaged in another call; B will hear a call waiting signal and has the option to terminate the current call. These two features will interact if a single subscriber subscribes to both of them. For example, if B subscribes to the two features and A calls B when B is engaged in a call to C, then it is not established whether A’s call will be forwarded, or whether B will receive a call waiting signal.

In practice, this interaction might be resolved by imposing some sort of precedence ordering on the features. The complexity of this situation becomes clear when one considers the potential of a system with hundreds of different possible features. Not only do interactions such as that of Example 2.1 have to be detected but appropriate resolution strategies must be formulated and implemented as well.

Although originally associated with the telephone system, feature interaction examples may be taken from any software system. Other instances in the literature (see [CM00, KB98, GR01]) include lift systems, email, IP telephony and home computing. Examples 2.2 and 2.3 below show interactions which might occur in a lift and an email system respectively.

**Example 2.2** Consider a lift system into which two features have been integrated: TTF ($\frac{2}{3}$-Full) and EF (Executive Floor). If the lift is more than two-thirds full, the TTF feature means that it will not respond to landing calls until existing passengers have left the lift, thereby reducing the lift’s load until it is less than two-thirds full. The EF feature means that the lift prioritises landing calls made from the executive floor. These two features interact because when the lift is two-thirds full, it is not specified whether the lift’s priority should be calls from the executive floor, or to unload current passengers.

**Example 2.3** If an email client program has some Decrypt feature which is executed prior to a Forward feature which forwards all messages to an alternative email address, email messages which should be encrypted will be sent to the forwarding address in their decrypted form.
The Feature Interaction Problem

For any system into which additional units of functionality may be integrated, it is possible to construct examples such as those above. Detection of interactions is a non-trivial process and it is difficult to envisage how this might be achieved automatically, without some kind of human intervention. However, in order to be exhaustive in detecting feature interactions, it is necessary to automate the process somehow. This is the paradox which lies at the heart of the feature interaction problem.

Many techniques and tools have been proposed in the feature interaction literature. However, it is difficult to compare and contrast these methods because they involve such a variety of approaches and there are few common points of reference, apart from the fact that they are based on the principal areas of integration, detection and resolution. Most methods focus on integration of features and detection of interactions between them. Other approaches are concerned with feature interaction resolution, and yet others with the prevention of feature interactions altogether. From an alternative point of view, some approaches are only applicable to the system design stage of a project lifecycle. This might be the case if the proposed architecture is too sophisticated to be applied to an existing system, e.g. [ZJ00]. On the other hand, a model-checking approach may be applied to an existing system because, by its very nature, model checking is based on tools which are used to model systems, e.g. [PR00, CM98]. When we review these varied approaches to the problem in Section 2.3, we will consider them in terms of the three principal areas as mentioned above.

2.2 Feature Integration

It is the operation of feature integration on which this research is based and we examine it in detail in this section. The topic is not often discussed in isolation because it is so closely bound up with feature interaction detection. However, we argue that it is important to consider it as an operation in its own right because an in-depth understanding of how feature integration works should lead to a clearer understanding of why certain interactions occur. It was this premise which led to the choice of this research topic, coupled with the compelling parallel of nonmonotonicity between feature integration and the logic of
theory change. First, we review existing methods of integration in Section 2.2.1, followed by a discussion of the nonmonotonic nature of the operation in Section 2.2.2.

2.2.1 Existing Methods of Feature Integration

Methods of feature integration always have an associated method of interaction detection (although this need not necessarily be the case vice versa). In fact, in some approaches (again we refer to [ZJ00] where the authors present their own virtual architecture called Distributed Feature Composition (DFC)), integration and interaction are so closely allied that it is difficult to distinguish between them. In DFC, integration of a feature basically involves the addition of a “feature box” to the system. Such a box may have any number of ports; its subscribers’ calls are simply routed via the relevant feature box and subscription information is updated. A similar idea (based on DFC) is presented in [Hal00] in which Hall outlines a method of integration and interaction detection for email systems.

In [Zua01], Zuanon provides another approach to integration which, like that of Zave and Jackson, can only really be used during system design. The reason for this is that composition of features always takes place before integration. Therefore, for a system which is dynamically updated all the time, such as a mobile phone, onto which new features may be downloaded as required, Zuanon’s approach is unlikely to be appropriate. With such an approach, the work involved in integrating a feature is likely to increase exponentially with each new feature, as it will have to be composed with all existing features prior to integration. With Zuanon’s method, a feature is specified in terms of three types of information: behaviour, requirements and behavioural patterns. Composition and integration are then defined for each of these pieces of information. A feature then “takes control” of a call as an automaton acting in parallel with the basic service automaton. Points of invocation and return are specified in the basic call automaton in a similar way to that of Turner in [Tur00].

In the majority of cases, the method of feature integration is very clear-cut. For example, when model-checking is used ([PR00, CM98]), integration usually involves a kind of macro construction which somehow modifies the system specification. More explicitly,
feature integration in the diagrammatic approach of Turner ([Tur00]), based on CRESS diagrams, involves the combination of two such diagrams. A feature specifies the numbers of the nodes to which it relates in the main system so that it can be “attached” correctly.

A predominantly logical approach is taken in both [GK94] and [A+99] where the system is specified by a set of axioms. A feature simply specifies which axioms are to be added, and which are to be modified. One could consider this in terms of an “add set” and a “delete set”.

Almost all of these methods of integration appear to be relatively straightforward to implement. The obvious reason for this is that each one is targeted at a particular approach. Feature integration needs to be as simple as possible because the latter stages of detection and resolution can become very complex and “messy”, and it is extremely undesirable to have to go back and revise the integration method later on. Another reason for the apparent simplicity of the methods is that there is probably a very limited number of possible methods of integration available for each approach. This means that those selected tend to seem obvious and are likely to be easily understood. Zuanon’s integration method is more complicated because his primary objective is to produce a strictly modular approach which enables composition of features prior to integration. For most approaches, explicit feature composition seems to be overlooked because it is assumed that, effectively, features are integrated into a system sequentially. Therefore, the integration method deals with integration of a single feature into the basic system and issues which arise as a result of integration of multiple features are dealt with at the stage of interaction detection.

The approach we have taken in our research differs from those presented above, because essentially we are treating the operation of feature integration from a theoretical perspective. The logical approaches of [GK94] and [A+99] still pose as solutions to the practical problem of feature interaction, whereas we have attempted to take a step back by applying a branch of logic to an existing method of feature integration in order that we may better understand this fundamental operation.
2.2.2 Issues of Nonmonotonicity

Nonmonotonicity is described in Chapter 1. Here, we use it to refer to the fact that existing properties of a system are violated on the integration of a feature.

Little work has been carried out in the field of feature interaction with regard to its nonmonotonic properties. However, [Vel95] is a seminal paper on the subject. It is cited in a large number of feature interaction publications, thus indicating that researchers in the area are obviously all too aware that nonmonotonicity is a serious issue. Perhaps the lack of active research on the subject is simply due to the fact that few individuals are knowledgeable enough about the two fields together. After all, feature interaction is predominantly a fairly practical field, whereas nonmonotonic reasoning is inherently theoretical.

Here, we summarise the findings as presented in [Vel95], in which Velthuijsen identifies three situations in feature interaction where nonmonotonicity is apparent. The three situations are described here:

1. **Nonmonotonic Extensions.** The main point here is one which has already been mentioned: integration of a feature into a system modifies original behaviour of the system such that the property of monotonicity is violated.

2. **Formulating Properties.** Formulation of system properties that help detect interactions - that is, interactions which are not known to exist already - is very difficult without attempting to analyse interactions manually beforehand. Moreover, once a feature has been integrated, it is often necessary for these properties to be reformulated, thus demonstrating the nonmonotonicity of this activity.

3. **The Qualification Problem.** The qualification problem deals with the fact that execution of an action often depends on so many preconditions that it is impossible to list them all, and it is also nigh on impossible to identify which of these preconditions are relevant. However, in the light of the nonmonotonic nature of feature integration, we need to know which system properties are relevant preconditions of the effects of a feature, in order that we are able to identify which of those effects
are altered when further features are integrated.

The thrust of Velthuijsen’s argument is that current Formal Description Techniques for feature interaction detection have to be improved if they are to be as useful as expected. In particular, they must incorporate the idea of nonmonotonicity in some way. He suggests that further investigation is required into how results from AI can be used to overcome this problem. Our research should go some way towards achieving this.

At this point, the method of Accorsi et al, as described in [A+00], should be mentioned. The authors cite Velthuijsen’s paper, and state the following: “... But the issue of nonmonotonicity ... has always been neglected. We believe that the non-monotonic nature of feature addition is one of the central themes in feature interaction.” They wanted to reconcile the fact that issues of nonmonotonicity appear to be central to feature interaction, with the computational complexity which logical approaches to nonmonotonicity seem to imply. Therefore, they use constraints as a natural way to model this nonmonotonicity. A constraint is a restriction on a space of possibilities. Mathematically, a constraint can limit the possible values of a variable. Constraint programming can be used to build systems using constraints. Accorsi et al represent a system by the set of its possible models defined using constraint rules. Features are then integrated by adding new constraints to the basic system. In this way old models may be pruned and new models generated. The nonmonotonic nature of this methodology is clear then, and although it is not the approach which we have explored in relation to the logic of theory change, it seems likely that it might yield similar positive results if we were to investigate it further.

2.3 Approaches to the Problem

The aim of this section is to put our research in context. We have already mentioned that many methods of integration and interaction detection have been proposed, and we have made reference to a number of them. In Section 2.3.1, we discuss the varying approaches to feature interaction detection which appear in the literature, in a similar way to that in which we discussed feature integration in Section 2.2.1. Then, in Section 2.3.2, we
summarise the approaches to which we have referred and which are most relevant to our research.

2.3.1 Existing Methods of Feature Interaction Detection

Many approaches to the feature interaction problem centre upon interaction detection. For example, as far as model-checking is concerned, integration is a fairly involved process but detection is straightforward isolation of models of the system which do not satisfy the specification, and this is an automated process. Methods in [PR00] (which uses the model-checker, SMV) and [CM98] (which uses SPIN) are typical examples of the model-checking approach. An interaction is detected if a path through the model is discovered which does not satisfy some set of specified properties.

Another means by which interactions are found is testing. With a software system, this involves carrying out a series of tests which verify that the system behaves as desired and expected. In this context, the failure of such tests is associated with identification of feature interactions. The methods in [Hal00] and [Zua01] use testing. Turner’s method in [Tur00] uses SDL or LOTOS to perform simulation and validation. This approach would appear to be a combination of model-checking and testing because it is an automated process which seems to be more exhaustive than testing but less exhaustive than model-checking. Issues of subjectivity are discussed below.

A logical approach is taken in [GK94], [A+99] and [A+00]. Effectively, logical deduction is used in the two former cases where a contradiction within the axiomatic description of the system indicates the presence of feature interaction. In the third of these logical approaches, constraint programming is used to model the system, so interactions are detected using query-based search whereby the existence of a model in which the query fails means that feature interaction has occurred.

The most unusual approach to interaction detection is exhibited in [ZJ00] where the authors focus on prevention and cure as opposed to detection. Discussion of interaction detection in the DFC architecture appears in [Zav01] where Zave distinguishes between her structural approach to detection, as opposed to the more common correctness approach.
She specifies that the former approach will only work if "...the feature specification language and composition operator constrain the ways in which features can interact. In this approach, the possible structural interactions are studied and classified. Then tools are developed to detect their presence in a feature set." Using this technique in a case study, all known interactions were detected.

A seemingly undesirable characteristic which manifests itself in the majority of approaches to interaction detection which we have seen is subjectivity. Thus far, no completely automated approach has become apparent; some degree of human judgement is always necessary. In the model-checking approach, specification of system requirements is subjective, as is selection of requisite tests for testing. Similarly, in constraint programming, the choice of query represents subjectivity, as does initial identification of the possible structural interactions in Zave’s structural approach. An entirely automated method of interaction detection only appears when logical deduction techniques are employed to find a contradiction in the axiomatic description of the system but, even then, construction of the axioms themselves is a subjective process. Presence of such subjectivity is undesirable but it is a ubiquitous part of the process for all the techniques considered here and, therefore, it is difficult to see quite how it may be eliminated altogether.

2.3.2 A Summary of Approaches

- **Accorsi et al ([A+00])** Constraints are used as a natural way to model nonmonotonicity. A constraint is a restriction on a space of possibilities. Mathematically, a constraint can limit the possible values of a variable. Constraint programming can be used to build systems using constraints. In this method, a telephone system is represented by the set of its possible models defined using constraint rules. Features are integrated by adding constraints to the basic system. In this way, old models are pruned and new models generated. For the implementation, stable models are employed which constitute a declarative semantics for logic programming. The intuition behind this approach is that it merges the advantages of logic programming knowledge base representation techniques with constraint programming. The **smod-**
els tool box is the specific implementation of stable models semantics which is used. Interactions are detected using exhaustive query-based testing.

- **Areces et al ([A+99])** Description logies are used to represent structured concepts and (possibly inter-related) objects where the latter may or may not satisfy some of the former. In this method, a description logic $\mathcal{FL}$ is proposed for describing feature interaction. The basic telephone system is represented by a knowledge base in $\mathcal{FL}$. Features are thought of as refinements of the knowledge base and integration is achieved by adding and deleting terminologies in the knowledge base. Interactions are then detected using a tableaux method for $\mathcal{FL}$. If there is an interaction, then a contradictory tableau will be constructed. Thus, feature interaction is defined as a satisfiability problem.

- **Calder, Miller ([CM01])** A telephone system is modelled using the Promela modelling language where system properties are LTL formulae. Features are integrated using a construct called the feature_lookup inline definition which is added to the Promela system specification. Interactions are detected using the Spin model-checker.

- **Gammelgaard, Kristensen ([GK94])** A logic is introduced to define a telephone system. It is specified using declarative transition rules which may be altered, deleted or added to when integrating a feature. Logical deduction detects interaction when two conflicting formulae are discovered which a state is expected to satisfy at once.

- **Hall ([Hal00])** A modular email system is specified using a specification modelling tool suite called ISAT which has been designed by the author. In the email architecture, a message travels from the sender to a receiver via the features to which the sender and the receiver subscribe. The system is simulated and interactions are detected by examining the outputs generated by the system from given input-event sequences. The input scenarios are selected based on human intuition and a formal test coverage tool.
The Feature Interaction Problem

- **Plath, Ryan ([PR00])** The SMV model checking system is used to specify both a telephone system and a lift system. Properties of the system are defined in CTL. A feature construct which overrides existing system behaviour is used to integrate a feature into a system. Interactions are then detected by model-checking.

- **Turner ([Tur00])** A basic telephone system is defined in terms of Cress diagrams which are directed cyclic graphs; features are also Cress diagrams but they are defined in relation to the POTS diagram where all the nodes are numbered. The diagrams are translated into SDL or LOTOS and interactions are detected using simulation.

- **Zave, Jackson ([ZJ00])** A network architecture called Distributed Feature Composition is created whereby each subscriber is represented by a line interface box and each feature is represented by a feature box. When a feature is integrated, a feature box is added to the architecture via which its subscribers’ calls will then be routed. Associated with each feature box are some operational data, and the routers are given data on feature subscriptions, feature precedences, dialling predicates and configuration. The authors of this method focus on prevention and cure of interactions, rather than detection.

- **Zuanon ([Zua01])** A basic telephone system is modelled by several automata. With each user is associated a single automaton called a logical telephone (LT) which is a behavioural specification of POTS and the features as viewed by its associated user. LT’s are synchronous and reactive. A feature is specified by three pieces of information: its behaviour, requirements and behavioural patterns. They are composed prior to integration and are represented by automata which run concurrently to the base system. Interactions are detected by means of testing where input generation is guided by the behavioural patterns of the features concerned. Behavioural patterns describe usages of a feature; they indicate methods available to the end user of activating or invoking a feature. Zuanon gives an example: “the Call Forward feature ... can be activated, deactivated and parameterised.” Features and the basic system are validated using their specified properties which are given
as requirements. The principle of local control is important in this method, hence the modular structure of the system.

2.4 SMV and the Feature Construct

It should be apparent, then, that feature interaction is a very broad area. It covers many topics and can be approached in a multitude of different ways. We have chosen to apply the theory of updates to Plath and Ryan’s approach to the feature interaction problem as presented in [PR00]. Their SMV feature construct is representative of a considerable proportion of feature interaction research because it involves feature integration and interaction detection, and it is generally classed as a software engineering approach. It is also a prime example to take because it has been successful in its application to both a telephone system and a lift system. In Section 2.4.1, we introduce the SMV language and in Section 2.4.2, we present Plath and Ryan’s SMV feature construct. In this section, we also introduce two examples of systems with associated features. These examples will then be used throughout the remainder of the thesis to illustrate the core results.

2.4.1 Introduction to SMV

SMV (Symbolic Model Verifier) [McM98] is a verification tool which takes as input a system description in the SMV language, and some formulae in the temporal logic, CTL. It outputs true or false for each of the CTL formulae depending upon whether or not they are satisfied by the system. If a formula is not satisfied, a trace is given to show circumstances in which this is the case.

The SMV language describes unlabelled nondeterministic finite state automata. It provides modularisation, and synchronous and asynchronous composition. For each variable (the type of which might be boolean, an enumeration, a finite range of integers or an array of these types), a set of possible initial values is declared and the next value is defined in terms of the values of variables in the current state. This is achieved by using case statements which are evaluated top to bottom and for which the cases are covering. A detailed introduction to SMV can be found in [McM93] and [McM98] and
The Feature Interaction Problem

Figure 2.1: A state transition diagram for the intruder deterrent light system.

CTL is described in [HR00].

**Example 2.4** The following program represents a simple intruder deterrent light which comes on from time to time for the duration of a single time point and then goes off again.

```plaintext
MODULE main
VAR
  switch : {on, off};
ASSIGN
  init(switch) := off;
  next(switch) := case
    switch = on : off;
    1 : {on, off};
  esac;
SPEC AG(switch = on -> AF switch = off)
```

The program in Example 2.4 has one variable called `switch` which is an enumerated type, the possible values of which are `on` and `off`. The initial value of `switch` is `off`. The case statements for the ‘next’ value of `switch` specify that if it is on, then it goes off; otherwise it could go on or off. Note that the default case (i.e. `else`) is specified by 1. This default behaviour is nondeterministic because it depends on the user’s settings, and perhaps the user will switch the light on manually. The last line of the program is a CTL formula which specifies that whenever the switch is on, then at some future time point it will go off; this property is, in fact, true of the system.

Figure 2.1 shows a transition system for the intruder deterrent light.

Now we consider a more complex example that has two binary variables.
Example 2.5 This example shows an SMV program which represents a level crossing. There are two variables: a barrier which can be up or down, and a boolean variable, train_coming. As one would expect, the barrier goes down when a train is coming, and stays up otherwise.

MODULE main
VAR
  train_coming : boolean;
  barrier : {up,down};
ASSIGN
  init(train_coming) := 0;
  init(barrier) := up;
  next(barrier) := case
      train_coming : down;
      1 : up;
    esac;
SPEC AG(train_coming -> AX barrier = down)

The CTL formula in Example 2.5 states that if a train is coming, the barrier will always be down in the next state. This property is satisfied by the system. Note that the behaviour of the train_coming variable is non-deterministic so it is not specified in SMV.

2.4.2 SFI: The SMV Feature Integrator

The feature construct for SMV involves extension of the syntax for SMV; a feature is specified using the extended syntax. During integration, such a feature is parsed and modifications are made to the base SMV program. It is not necessary to define the precise syntax of the feature construct here. We are concerned with IMPOSE statements of the feature which change variable values. Specifically, an IMPOSE statement of the form:

IF.cond THEN IMPOSE \text{next}(x) := expr;

means that assignments such as \text{next}(x) := oldexpr; have oldexpr replaced by:
case
  cond : expr;
  1  : oldexpr;
esac

Whenever \texttt{cond} is true, the value \texttt{expr} is imposed on \texttt{next}(x). A statement of the form \texttt{IMPOSE next(x):= expr} which is not guarded by a condition has the \texttt{case} statements omitted and \texttt{oldexpr} is replaced by \texttt{expr} directly.

\textbf{Example 2.6} We introduce a feature to be integrated into the program in Example 2.4. We want the intruder deterrent light to be on constantly in the evening between the hours of 8pm and 11pm, say. \texttt{evening_hours} is a boolean variable which is true during these hours.

\texttt{IF evening_hours THEN}
\begin{verbatim}
  IMPOSE next(switch) := on;
\end{verbatim}

\textbf{Example 2.7} The program of Example 2.4 into which the feature of Example 2.6 has been integrated.

\texttt{MODULE main}
\begin{verbatim}
VAR
  switch : \{on,off\};
  evening_hours : boolean;
ASSIGN
  init(switch) := off;
  next(switch) := case
    evening_hours : on;
    1  : case
      switch = on : off;
      1  : \{on,off\}
    esac;
esac;
SPEC A\(G\) (\(\text{switch} = \text{on} \rightarrow \text{AF switch} = \text{off}\))

In the resulting program, the behaviour of the \text{evening hours} variable is unspecified, so it will assume values nondeterministically. In this case, the CTL formula is not satisfied because if \text{evening hours} was always true (we know that it will not be, but as far as SMV is concerned, it could be), then there is an execution of the program in which \text{switch} has the value \text{on} forever.

\textbf{Example 2.8} Now we consider a feature to be integrated into the level crossing program of Example 2.5. In an emergency, we may wish the barrier to remain down, whether a train is coming or not, so we need a new \text{emergency} variable.

\textbf{IF} \text{emergency} \textbf{THEN}
\begin{quote}
\text{IMPOSE next(barrier) := down;}
\end{quote}

\textbf{Example 2.9} The program of Example 2.5 into which the feature of Example 2.8 has been integrated.

\textbf{MODULE} \text{main}
\textbf{VAR}
\begin{quote}
\text{train_coming} : \text{boolean};
\text{barrier} : \{\text{up, down}\};
\text{emergency} : \text{boolean};
\end{quote}
\textbf{ASSIGN}
\begin{quote}
\text{init(train_coming) := 0;}
\text{init(barrier) := up;}
\text{next(barrier) := case}
\begin{quote}
\text{\hspace{1em}emergency : down;}
\text{\hspace{1em}1 : case}
\begin{quote}
\text{\hspace{2em}train_coming : down;}
\text{\hspace{2em}1 : up}
\end{quote}
sac;
\end{quote}
esac;
SPEC AG(train_coming -> AX barrier = down)

A full explanation of the work summarised in this section can be found in [PR00].

Following an introduction to the logic of theory change in Chapter 3, we will see how SMV and the feature construct relate to the update operation in Chapters 4 and 5.
Chapter 3

The Logic of Theory Change

The logic of theory change is used to reason about a knowledge base (KB) which, on the acquisition of new information, needs to be revised in some way in order to incorporate the new information whilst maintaining consistency. The operations of theory change considered here are defined over propositional logic. *Belief revision* is the most widely known such operation. In 1985, Alchourrón, Gärdenfors and Makinson proposed eight rationality postulates which they argue should be satisfied by any operator of belief revision; they are known as the AGM postulates [AGM85]. The motivation behind these postulates is that, in incorporating a new belief into a knowledge base, they should result in minimal change of the knowledge base. It is actually the operation of *update*, a variation on belief revision, which we relate to feature integration in this thesis. Katsuno and Mendelzon presented update in [KM92] and they defined it in terms of eight rationality postulates of its own. Belief revision and update are presented fully in Sections 3.1 and 3.2 respectively.

Apart from belief revision and update, various other operations of theory change have been proposed. These include *screened revision*, in which a core set of beliefs is specified which may not be violated on revision of the knowledge base, *local change*, an operation defined in terms of belief bases which may contain inconsistencies, and *iterated revision*. We introduce these three further operations in Section 3.3 as they all merit consideration with regard to feature integration.

Finally, in Section 3.4, theory change is discussed with regard to *nonmonotonic reason-
In Chapter 1, we referred to the fact that, in studying the relationship between theory change and feature integration, we are attempting to draw upon the nonmonotonicity of feature integration. The relationship between theory change and nonmonotonic reasoning is well-known and was formalised over a decade ago by Makinson and Gärdenfors in [MG91]; their result is described here.

3.1 Belief Revision

The operation of belief revision is performed upon a knowledge base which takes the form of a belief set. The knowledge base has to be revised when new information about the world is acquired. A belief set is a set of propositional logic sentences (or formulae\(^1\)) which represent beliefs about the world. If the new information is consistent with the current belief set, then it may simply be expanded by the new information, otherwise it must be modified in order to incorporate the new information. The principal definition of belief revision was introduced by Alchourrón, Gärdenfors and Makinson in 1985. They proposed eight rationality postulates which a belief revision operator should satisfy; potentially, there are a number of belief revision operators which satisfy these postulates. Following some preliminaries, the AGM postulates are presented below. Belief revision may also be considered within a semantic framework called the preferential models approach (PMA), which is also presented here.

To begin with, here is a concrete example of belief revision taken from [Gär92].

**Example 3.1** Consider the following knowledge base and new information:

**KNOWLEDGE BASE**

\[\alpha: \text{All European swans are white.}\]
\[\beta: \text{The bird caught in the trap is a swan.}\]
\[\gamma: \text{The bird caught in the trap comes from Sweden.}\]
\[\delta: \text{Sweden is part of Europe.}\]

\(^1\)The terms *sentence* and *formula* are synonymous in the context of belief sets and propositional logic, and may be used interchangeably. See Chapter 1 of [Gär92]. We use both terms throughout this thesis.
NEW INFORMATION

ε : The bird caught in the trap is black.

When we try to revise the knowledge base by the new information, we find that ε conflicts with a fact which is derivable from \{α, β, γ, δ\}: that the bird caught in the trap is white. In resolving this conflict, one of the facts in the original knowledge base must be retracted, and this is what happens in order for the revision to be successful.

3.1.1 Preliminaries

We assume a propositional logic language \(\mathcal{L}\) which consists of a set of atomic propositions \(\text{Atoms}\), and a set of logical connectives.

**Definition 3.1 (Formulae)** The BNF syntax of a formula \(\phi\) over \(\mathcal{L}\) is defined as follows:

\[
φ ::= p \mid \neg φ \mid φ_1 ∧ φ_2 \mid φ_1 ∨ φ_2 \mid φ_1 → φ_2 \mid φ_1 ↔ φ_2
\]

where \(p ∈ \text{Atoms}\).

**Definition 3.2 (Models)** A model \(m\) is an assignment of truth values to each \(x ∈ \text{Atoms}\). \(\mathcal{M}\) denotes the set of all models over \(\mathcal{L}\). Model \(m\) satisfies formula \(φ\), that is \(m \models \phi\), if \(φ\) is true in \(m\). The set of models associated with a formula \(φ\), \(\text{Mod}(φ)\), is defined as follows:

\[
\text{Mod}(φ) = \{m \mid m \models \phi\}
\]

The set of models associated with a set of formulae \(F\), \(\text{Mod}(F)\), is defined as follows:

\[
\text{Mod}(F) = \bigcap_{φ∈F} \text{Mod}(φ)
\]

**Definition 3.3 (Entailment)** A set of sentences \(F\) over \(\mathcal{L}\) entails a formula \(φ\), that is \(F \models \phi\) if and only if

\[
\forall m ∈ \text{Mod}(F) . m \models \phi
\]
**Definition 3.4 (Provability)** From a set of sentences $F$, we can prove a formula $\phi$, that is $F \vdash \phi$ if propositional logic proof theory may be used to conclude $\phi$ from $F$. The set of all logical consequences, or logical closure, of $F$ is denoted by $\text{Cn}(F)$ and is formally defined as follows:

$$\text{Cn}(F) = \{ \phi \mid F \vdash \phi \}$$

Belief revision, as presented by Alchourrón, Gärdenfors and Makinson (AGM) in [AGM85], is defined over belief sets.

**Definition 3.5 (Belief Set)** A belief set $K$ contains a set of formulae over $\mathcal{L}$ and is closed under logical consequence.

**Definition 3.6 (The Inconsistent Belief Set)**

$$K_{\bot} = \text{Cn}(\{ \bot \})$$

is the unique inconsistent belief set. It contains every possible sentence over $\mathcal{L}$.

**Definition 3.7 (Expansion)** We expand a belief set $K$ when we add a sentence $\phi$ to it and close under logical consequence:

$$K + \phi = \text{Cn}(K \cup \{ \phi \})$$

Throughout this section, $K$ is a belief set and $\phi$ is a sentence over $\mathcal{L}$.

### 3.1.2 The AGM Postulates for Revision

In this section, we present the eight AGM rationality postulates for belief revision. Each postulate appears below with a brief explanation. They were introduced by Alchourrón, Gärdenfors and Makinson (AGM) in [AGM85]. These postulates represent the logical properties which we expect an operator of revision to satisfy. However, they do not uniquely characterise belief revision; various * operators might be defined, all of which satisfy the AGM postulates. $K * \phi$ denotes belief set $K$ revised by formula $\phi$. Note that the rationality postulates presented below represent a simplified version of those given in [Gär92].
(K*1) For any sentence $\phi$ and any belief set $K$, $K \ast \phi$ is a belief set.

The first postulate ensures that the output is always a set of logical sentences closed under logical consequence.

(K*2) $\phi \in K \ast \phi$

This is known as the axiom of success and states that the input sentence $\phi$ should be contained in the output belief set.

(K*3) $K \ast \phi \subseteq K + \phi$

(K*4) If $\neg \phi \not\in K$, then $K + \phi \subseteq K \ast \phi$

(K*3) and (K*4) constitute a pair which specifies that if $\phi$ is inconsistent with $K$ then the output belief set should be a subset of $K$ expanded by $\phi$. Otherwise, $\phi$ should simply be added to $K$ (i.e. expansion should indeed occur). These two postulates mean that the revision operation will reduce to expansion whenever possible.

(K*5) $K \ast \phi = K\bot$ if and only if $\vdash \neg \phi$

As long as $\phi$ is consistent, the output belief set should always be consistent. However, (K*5) specifies that if $\phi$ is logically impossible, then the output set is the unique inconsistent belief set, $K\bot$.

(K*6) If $\vdash \phi \leftrightarrow \psi$, then $K \ast \phi = K \ast \psi$

The sixth postulate states that revision of $K$ by either one of two logically equivalent sentences will result in an identical output belief set. This ensures that the revision operation is defined on a semantic level, as opposed to a syntactic one.

(K*7) $K \ast (\phi \land \psi) \subseteq (K \ast \phi) + \psi$

(K*8) If $\neg \psi \not\in K \ast \phi$, then $(K \ast \phi) + \psi \subseteq K \ast (\phi \land \psi)$

The first six postulates are concerned with the basic requirements of the revision operation, and the last two relate to composite revisions, e.g. $K \ast (\phi \land \psi)$. The idea is that, as long as $\psi$ does not contradict the belief set produced by $K \ast \phi$, then
Figure 3.1: The operation $K \ast \phi$ selects the closest models of $\phi$ with respect to the models of $K$.

$K \ast (\phi \land \psi)$ should be achieved by expansion of $K \ast \phi$ by $\psi$. Otherwise, $K \ast (\phi \land \psi)$ should be a subset of the expanded belief set.

### 3.1.3 The Preferential Models Approach

The preferential models approach (PMA) enables us to consider belief revision from a model-theoretic, as opposed to a proof-theoretic, point of view. The idea is that, in revising $K$ by $\phi$, the minimal models of $\phi$ are selected with respect to the models of $K$. This is illustrated in Figure 3.1.

By rephrasing the AGM postulates in terms of finite covers for possibly infinite belief sets\(^2\), Katsuno and Mendelzon proved a representation theorem for belief revision in [KM89]. We show the rephrased rationality postulates (R1)-(R6) and two necessary definitions, followed by their theorem. $\circ$ is the altered $\ast$ operator as defined by the rephrased

\(^2\)In this context, a ‘finite cover’ is a propositional logic formula $\psi$, representing the belief set $K$ with which it is associated. When $K$ is finite, $\psi$ can simply be constructed by taking the conjunction of the formulae in $K$. However, we really need these ‘finite covers’ when $K$ is infinite, in which case $\psi$ represents some finite translation of $K$. The reason we need to eliminate the potentially infinite nature of belief sets is that we will be reasoning about sets of models of belief sets and, in particular, we wish to impose orderings upon these models in order to find those models which are minimal. If a belief set is infinite, then it will have an infinite set of models associated with it and, in that case, a minimal set of models may not exist.
rationality postulates. \( \phi, \psi \) and \( \mu \) represent formulae over the propositional logic language \( \mathcal{L} \), as defined in Section 3.1.1.

(R1) \( \psi \circ \mu \) implies \( \mu \).

(R2) If \( \psi \land \mu \) is satisfiable then \( \psi \circ \mu \Leftrightarrow \psi \land \mu \).

(R3) If \( \mu \) is satisfiable then \( \psi \circ \mu \) is also satisfiable.

(R4) If \( \psi_1 \Leftrightarrow \psi_2 \) and \( \mu_1 \Leftrightarrow \mu_2 \) then \( \psi_1 \circ \mu_1 \Leftrightarrow \psi_2 \circ \mu_2 \).

(R5) \( (\psi \circ \mu) \land \phi \) implies \( \psi \circ (\mu \land \phi) \).

(R6) If \( (\psi \circ \mu) \land \phi \) is satisfiable then \( \psi \circ (\mu \land \phi) \) implies \( (\psi \circ \mu) \land \phi \).

We define a preorder between models.

**Definition 3.8 (Preorder)** A preorder \( \leq \) on \( \mathcal{M} \) is a reflexive and transitive relation. We define \( < \) in terms of \( \leq \) as follows:

\[
\forall m, n \in \mathcal{M} \cdot m < n \overset{\text{def}}{=} m \leq n \land n \not\leq m
\]

A preorder is total iff

\[
\forall m, n \in \mathcal{M} \cdot m \leq n \lor n \leq m
\]

A preorder is partial iff

\[
\exists m, n \in \mathcal{M} \cdot m \not\leq n \land n \not\leq m
\]

**Definition 3.9 (Faithful Assignment)** Consider a function that assigns to each propositional formula \( \psi \) a preorder \( \leq_\psi \) over \( \mathcal{M} \). We say this assignment is faithful if the following three conditions hold:

1. If \( m, n \in \text{Mod}(\psi) \) then \( m <_\psi n \) does not hold.

2. If \( m \in \text{Mod}(\psi) \) and \( n \notin \text{Mod}(\psi) \) then \( m <_\psi n \) holds.

3. If \( \psi \leftrightarrow \phi \), then \( \leq_\psi = \leq_\phi \).
The Logic of Theory Change

Now we are ready to introduce the representation theorem, Theorem 3.1.

**Theorem 3.1** Revision operator \( \circ \) satisfies Conditions (R1)-(R6) if and only if there exists a faithful assignment that maps each knowledge base \( \psi \) to a total pre-order \( \leq_\psi \) such that \( \text{Mod}(\psi \circ \mu) = \text{Min}(\text{Mod}(\mu), \leq_\psi) \).

An example of a \( \leq \)-ordering on models which yields a faithful assignment is that specified by Dalal in [Dal88]. His ordering is shown here, where \( x, y, m \in \mathcal{M} \), \( \psi \) is a formula and \( p \in \text{Atoms} \). First, we define a measure of ‘distance’ between two models, \( x \) and \( y \):

\[
d(x, y) = \{ p \mid x \vdash p \text{ and } y \not\vdash p, \text{ or } x \not\vdash p \text{ and } y \vdash p \}
\]

Now we can define a measure of ‘distance’ between a set of models for a formula \( \psi \) and a single model \( y \):

\[
d(\psi, y) = \text{Min}_{x \in \text{Mod}(\psi)} |d(x, y)|
\]

Finally, we define a \( \leq_\psi \) ordering on models:

\[
x \leq_\psi y \iff d(\psi, x) \leq d(\psi, y)
\]

Various other orderings on models have been proposed in the literature but, for belief revision, it is only Dalal’s which satisfies (R1)-(R6).

### 3.2 Update

It is the update operation, a variation on belief revision, which we prove is related to feature integration. Katsuno and Mendelzon presented update in [KM92] and they defined it in terms of eight rationality postulates of its own. They claim that the difference between update and revision lies in the fact that revision models changing beliefs about a static world, whereas update models a changing world.

In this section, we present the eight rationality postulates for update followed by Katsuno and Mendelzon’s representation theorem for update. Then we discuss the difference between update and revision in more detail and justify why it is the update operation which we have chosen to relate to feature integration.
3.2.1 Katsuno and Mendelzon’s Rationality Postulates for Update

The postulates, (U1)-(U8) are given below with explanations. The operation is defined over propositional logic and $\psi \circ \mu$ denotes the result of updating a formula $\psi$ by a formula $\mu$.

(U1) $\psi \circ \mu$ implies $\mu$.

The update should successfully incorporate the new information into the knowledge base.

(U2) If $\psi$ implies $\mu$ then $\psi \circ \mu$ is equivalent to $\psi$.

If the new information is already implied by the knowledge base, then on update it should remain unchanged.

(U3) If both $\psi$ and $\mu$ are satisfiable then $\psi \circ \mu$ is also satisfiable.

If both the knowledge base and the new information are satisfiable, then the updated knowledge base should also be satisfiable.

(U4) If $\psi_1 \leftrightarrow \psi_2$ and $\mu_1 \leftrightarrow \mu_2$ then $\psi_1 \circ \mu_1 \leftrightarrow \psi_2 \circ \mu_2$.

Equivalent knowledge bases and new information should result in equivalent updated knowledge bases.

(U5) $(\psi \circ \mu) \land \phi$ implies $\psi \circ (\mu \land \phi)$.

This postulate ensures that update is carried out with minimal change.

(U6) If $\psi \circ \mu_1$ implies $\mu_2$ and $\psi \circ \mu_2$ implies $\mu_1$ then $\psi \circ \mu_1 \leftrightarrow \psi \circ \mu_2$.

For two sentences, $\mu_1$ and $\mu_2$, if updating a knowledge base with one guarantees the other, then the two updates have the same effect.

(U7) If $\psi$ is complete then $(\psi \circ \mu_1) \land (\psi \circ \mu_2)$ implies $\psi \circ (\mu_1 \lor \mu_2)$.

For a complete knowledge base, if some possible world results from the update of $\mu_1$ and also from the update of $\mu_2$, then it must also result from the update of $\mu_1 \lor \mu_2$.
(U8) \((\psi_1 \lor \psi_2) \circ \mu \leftrightarrow (\psi_1 \circ \mu) \lor (\psi_2 \circ \mu)\).

This is known as the disjunction rule and guarantees that each possible world of the knowledge base is given independent consideration.

### 3.2.2 Representation Theorem

The (U1)-(U8) axioms claim to characterise precisely what it is to be an update operator. This claim is in part justified by a ‘representation theorem’ proved by Katsuno and Mendelzon in [KM92] which says that any update operator which satisfies (U1)-(U8) may be characterised by an ordering on models. This is similar to Katsuno and Mendelzon’s representation theorem for belief revision which we described in Section 3.1.3, only for update there is no need to rephrase the rationality postulates because they are already defined in terms of finite propositional logic formulae, as opposed to potentially infinite belief sets. Given three models, \(x\), \(y\) and \(m\), we can define exactly what is meant by \(x \leq_m y\), or ‘\(x\) is at least as close to \(m\) as \(y\) is.’ Katsuno and Mendelzon’s theorem means that we may define \(\circ\) operators which satisfy (U1)-(U8) by defining such orderings.

**Example 3.2** Winslett [Win88] and Dalal [Dal88] defined \(x \leq_m y\) as follows:

(Winslett, 1988): \(x \leq_m y \leftrightarrow d(x, m) \subseteq d(y, m)\)

(Dalal, 1988): \(x \leq_m y \leftrightarrow |d(x, m)| \leq |d(y, m)|\)

where \(d(m_1, m_2)\) is a measure of ‘distance’ between \(m_1\) and \(m_2\) as before, e.g.

\[ d(m_1, m_2) = \{p \mid m_1 \models p \text{ and } m_2 \not\models p, \text{or } m_1 \not\models p \text{ and } m_2 \models p\} \]

where \(p\) is an atomic proposition. The motivation behind this definition is that the distance between \(m_1\) and \(m_2\) is large if they disagree on a lot of propositions.

The representation theorem defines the connection between the update operator and an ordering on models. It states that a \(\psi \circ \mu\) operation satisfies (U1)-(U8) if and only if there exists some \(\leq\)-ordering on models such that:

\[
\text{Mod}(\psi \circ \mu) = \bigcup_{m \in \text{Mod}(\psi)} \text{Min}_{\leq_m} (\text{Mod}(\mu))
\]
where $\text{Mod}(\phi)$ represents the set of models of formula $\phi$ and $\text{Min}_{\leq m}(\text{Mod}(\phi))$ represents the set of minimal models of $\phi$ with respect to the model $m$. Winslett and Dalal's $\leq$-orderings, as defined above, may both be used to define a $\odot$ operator.

In other words, the $\odot$ operator selects the set of minimal models (according to some ordering) of $\mu$ with respect to each model of $\psi$ and takes the union of these sets\(^3\). In terms of programs and features, this representation theorem gives us the concept of ordering models of a feature with respect to a program. In the future this might give us a means by which we can order features in terms of their ‘closeness’ to the base system. This possibility is discussed in more detail in Chapter 5.

### 3.2.3 The Difference between Update and Revision

Recall that Katsuno and Mendelzon claim that the difference between update and revision lies in the fact that revision models changing beliefs about a static world, whereas update models a changing world. Intuitively, then, the reason why we have selected update and not belief revision is that the ‘world’ we are revising is a reactive system and, in adding a feature to it, we are changing the world itself, as opposed to changing beliefs about a world which is static.

There are several salient differences between update and revision apart from the general one given above. These differences are explored in detail in [KM92], but we summarise them here. First of all, let us take a model-theoretic point of view, where $\psi$ is the knowledge base and $\mu$ is the sentence representing the new information. Revision selects the models of $\mu$ which are closest to the set of models of $\psi$. Update selects the set of closest models of $\mu$ for each model of $\psi$ and then takes the union of these sets. Closeness between models is defined by some ordering relationship on models. The difference here lies in the fact that revision is a setwise operation and update is pointwise. For revision, this behaviour means that some possible models of $\psi$ are, in effect, ruled out once the operation has taken place. This is rational because revision models changing beliefs about a static world and new information may indeed lead to the conclusion that worlds which

\(^3\)See Appendix B for an amusingly modified diagram which illustrates this.
were once deemed to be possible are in fact impossible. However, for update, all possible worlds must always be considered because update models a changing world. One model of $\psi$ is a model of the real world, but it is not possible to ascertain which one, so it is necessary to find the models of $\mu$ which are closest to each of them.

An important difference between the two operations is implied by postulate (U2). If a knowledge base is inconsistent, revision guarantees that the resulting knowledge base will be consistent, whereas, with update, an inconsistency can never be eliminated. Again, this makes sense in terms of our initial description of the difference between the operations. In revision, contradictory beliefs in a knowledge base can be overcome by a new piece of information which resolves the inconsistency. On the other hand, in update, an inconsistent knowledge base means that there are no possible worlds, so we do not have a real world to update. (U2) also highlights the fact that new information can simply be added to the knowledge base if it is already implied by existing information in the knowledge base. This again contrasts with revision which makes a stronger assertion: that as long as new information is consistent with the knowledge base, it can simply be added.

**Example 3.3** (based on an example in [Gär92]). Consider a situation in which either a book or a magazine is on a table: $b \lor m$. A robot is then ordered to put the book on the floor so we learn that $\neg b$. In revision, the resulting knowledge base contains $\neg b \land m$, whereas in update, we get $\neg b$.

The contrasting results in the example are easily explained. Revision models changing beliefs about a static world, so the new worlds are simply as close as possible to what we believe to be the case, whilst satisfying $\neg b$. However, in terms of update, this new information causes us to reflect upon the fact that the world itself could have changed; we are not certain that $b \lor m$ will still be true in the real world.
3.3 Other Operations of Theory Change

In this section, three further operations of theory change are introduced. Although they are not explored in relation to feature integration in this thesis, they all merit consideration in terms of possible future work.

3.3.1 Screened Revision

In belief revision, a new belief is to be added to a knowledge base and it is this objective which takes priority in the sense that, except for the fact that change to the KB must be minimal, there are no other factors to be taken into consideration. However, it is argued in the literature, specifically in [Sch98] and [Mak98], that this prioritisation of new information is not always either desirable or realistic. For example, we may have a set of basic beliefs in our KB which we never wish to violate. In this case, it is necessary to reject a new piece of information which conflicts with these basic beliefs, as opposed to retracting a subset of them in order to revise successfully by the new information. Another example is that, in performing the operation $K * \phi$, we may wish to consider not only the minimal models of $\phi$ with respect to $K$, but also the minimal models of $K$ with respect to $\phi$. These two examples summarise the ideas behind the two approaches to belief revision which will be defined in this section. First we will look at Makinson's approach to what he calls screened revision in [Mak98] and then we will examine the method proposed by Schlechta in [Sch98].

In his Platonic dialogue, Makinson gives the following initial definition of screened revision:

**Definition 3.10 (Plain Screened Revision)**

\[ K \#_A \alpha = \begin{cases} 
K *_A \alpha & \text{if } \alpha \text{ is consistent with } A \cap K \\
K & \text{otherwise}
\end{cases} \]

where $K *_A \alpha$ is belief revision of $K$ by $\alpha$ such that $A$ is a set of sentences which are protected.

Now we can give a model-theoretic definition of screened revision.
The Logic of Theory Change

Definition 3.11 (Model-Theoretic Screened Revision)

\[
\text{Mod}(K \#_{A} \alpha) = \begin{cases} 
\text{Min}_{K}(\text{Mod}(\alpha) \cap (\text{Mod}(A) \cup \text{Mod}(K))) & \text{if } \text{Mod}(\alpha) \cap (\text{Mod}(A) \cup \text{Mod}(K)) \neq \emptyset \\
\text{Mod}(K) & \text{otherwise}
\end{cases}
\]

There are actually three possible scenarios which might arise in model-theoretic screened revision:

1. \( \text{Mod}(\alpha) \cap (\text{Mod}(A) \cup \text{Mod}(K)) = \emptyset \Rightarrow \text{Mod}(K \#_{A} \alpha) = \text{Mod}(K) \)

2. \( \text{Mod}(\alpha) \cap \text{Mod}(K) \neq \emptyset \Rightarrow \text{Mod}(K \#_{A} \alpha) = \text{Mod}(K) \cap \text{Mod}(\alpha) \)

3. \( \text{Mod}(\alpha) \cap \text{Mod}(K) = \emptyset \Rightarrow \text{Mod}(K \#_{A} \alpha) = \text{Min}_{K}(\text{Mod}(A) \cap \text{Mod}(\alpha)) \)

In Figure 3.2, the shaded area shows the possible space in which models of screened revision might be found for cases 2 and 3 above. There are three sections of the shaded area, and they will not necessarily all contain models in both cases.

However, Makinson decided that this definition was too simplistic because the set \( A \cap K \) is always preserved, no matter what the new information \( \alpha \) may be. His solution was to impose a \( \prec \)-ordering on sentences of the language, such that \( \alpha \prec \beta \) means “\( \alpha \) is less credible than \( \beta \)” Then we have a more general definition of screened revision.

Definition 3.12 (Relationally Screened Revision)

\[
K \#_{<} \alpha = \begin{cases} 
K *_{\{\beta \mid \alpha \prec \beta\}} \alpha & \text{if } \alpha \text{ is consistent with } \{\beta \mid \alpha \prec \beta\} \cap K \\
K & \text{otherwise}
\end{cases}
\]

In [Sch98], the author begins with the same objective as had Makinson in [Mak98]: to find a revision operator which does not blindly incorporate new information into the knowledge base regardless of what it is. Schlecht does indeed present a more discriminating revision operator but it is quite different from that of Makinson. The main difference is that for a knowledge base \( K \), and a sentence \( \phi \), Makinson’s operator either results in models of \( K \) or models of \( K * \phi \), whereas Schlecht’s operation results in both models of \( K \) and models of \( K * \phi \). For this new operation, \( K \otimes \phi \), Schlecht says that the resultant
knowledge base contains ‘ . . . formulas valid in $[K]$ or $\phi$ with minimal distance among all pairs of $[K]$ and $\phi$ models.’

Let us look at Schlehta’s model-theoretic definitions of $K \ast \phi$ (prioritised belief revision) and $K \otimes \phi$ (non-prioritised belief revision).

**Definition 3.13 (Model-Theoretic Non-Prioritised Belief Revision)**

$$A \mid B = \{ b \mid b \in B \land \exists a_b \in A \land \forall a' \in A, b' \in B . \ d(a_b, b) \leq d(a', b') \}$$

$$A \uparrow B = \{ a, b \mid a \in A \land b \in B \land \forall a' \in A, b' \in B . \ d(a, b) \leq d(a', b') \}$$

where $\mathcal{L}$ is a logic, $M_\mathcal{L}$ is the set of models of $\mathcal{L}$, $d$ is a distance on $M_\mathcal{L}$ and $A, B \subseteq M_\mathcal{L}$.

Now we can look at his definitions of the two types of revision in terms of belief sets.

**Definition 3.14 (Non-Prioritised Belief Revision)**

$$K \ast \phi = Th(\text{Mod}(K) \mid \text{Mod}(\phi))$$
$K \otimes \phi = Th(\text{Mod}(K) \uparrow \text{Mod}(\phi))$

where $Th(X) = \{ \psi \mid (\psi \in \mathcal{L}) \land (X \subseteq M_\mathcal{L}) \land (\exists x \in X. x \models \psi) \}$.

Figure 3.3 illustrates the operation of non-prioritised belief revision.

It is clear that $\text{Mod}(K \otimes \phi) = \text{Mod}(K \ast \phi) \cup \text{Mod}(\phi \ast K)$ but Schlechta gives the following reason for deciding not to base the non-prioritised case directly on the prioritised one: ‘Non-prioritized Theory Revision is not yet a very established theory, and intuitions might change. At this stage, it might be preferable not to mix non-prioritized and prioritized Theory Revision.’

Both researchers then went on to investigate the logical properties of the new operations. We do not describe these results here as we are simply looking to give an idea of what the operations entail.

Intuitively, screened revision seems relevant to the operation of feature integration because, on integration of a feature into a base system, we may wish to preserve a core set of properties of the system. In fact, one might think that this is almost certain to be the case in the context of systems and features. However, we leave this line of enquiry to future research because, unlike update and belief revision, screened revision is a relatively new area and we know little of its logical properties.
\[3.3.2 \text{ Local Change}\]

Another topic which we have investigated briefly is that of local change. So far in this chapter, we have considered belief revision in terms of operations on belief sets which are closed under logical consequence. If an inconsistency is found in a belief set, it is corrupted and formally contains every possible belief: $K_\perp$. In the field of local change, belief bases are employed instead. They are not closed under logical consequence and so the beliefs in a belief base are deemed to be more ‘basic’ beliefs, from which further beliefs may be derived through logical closure. In a belief base, then, there may be inconsistencies; we have two new operations, consolidation and semi-revision, which may be performed upon it. Consolidation makes an inconsistent belief base consistent, and semi-revision is an operation which may either accept or reject the input sentence (a similar idea to non-prioritised belief revision as in Section 3.3.1). In [WH00], it is argued that actual agents in the real world often have local inconsistencies in their belief states which do not induce us to believe in everything. They go on to define several local operations on belief bases. The belief base is divided into compartments in the sense that each sentence or set of sentences has a compartment ‘around’ it. For a belief base $B$ and a set of sentences $A \subseteq B$, the $A$-compartment of $B$, $c(A, B) = \bigcup_{\alpha \in A} c(\alpha, B)$. The compartments in a belief base depend on the underlying logic, and so they may be defined using kernels. Given an inference operator $C$ and a sentence $\alpha$, $X \subseteq B$ is an $\alpha$-kernel iff it is a minimal set implying $\alpha$. Wassermann and Hansson go on to study the logical properties of the local revision operators which they define in [WH00].

The motivation behind our interest in the subject of local change is that it could be applicable to feature integration. Intuitively, it could well be the case that a feature affects only one particular ‘compartment’ of the system. As long as this compartment is well-defined, it should be possible to consider it in isolation when we integrate the corresponding feature. On the other hand, it does not seem desirable for it to be possible for the system to contain inconsistencies. Again, this subject falls outside the scope of the research in this thesis but it constitutes a possible area for future research.
3.3.3 Iterated Belief Revision

We have conducted little research into iterated revision. It seems interesting because feature integration is inherently an iterative operation. That is, in order to be able to study the interaction between two features, one feature has to be integrated into the system followed by the other, unless an alternative method is used under which the features are composed prior to integration. In [DP94], the authors argue that the AGM postulates cannot properly regulate iterated belief revision. It is a serious conjecture and ought to be examined at some point in the future.

3.4 Nonmonotonic Reasoning

In this section, we introduce nonmonotonic reasoning (NMR). Recall that it was the inherent nonmonotonicity of both feature integration and belief revision which led us to investigate the relationship between the two topics.

In Section 3.4.1, we give an example and define nonmonotonic logic in terms of preferential models. Then, in Section 3.4.2, we present the logical properties of nonmonotonic reasoning. Finally, we give a short summary of some research into the formal relationship between NMR and belief revision in Section 3.4.3.

3.4.1 Preliminaries

In classical logic, the following inference rule, that of monotonicity, is trivial:

\[ \phi \vdash \psi \]
\[ \phi, \chi \vdash \psi \]

where \( \phi \), \( \psi \) and \( \chi \) are sets of logical formulae. If a set of conclusions \( (\psi) \) may be derived from a particular set of premises \( (\phi) \), then no matter what we add to the set of premises \( (\chi) \), we will still be able to derive the same set of (possibly extended) conclusions. However, in nonmonotonic logic this axiom does not necessarily hold.

In a sense, nonmonotonic reasoning is closer to human reasoning than classical logic and it did indeed begin as an attempt to model such reasoning. Let us consider a canonical example in the field of NMR.
Example 3.4 We know that, normally, birds fly and that penguins cannot fly. Now we are told that Tweety is a bird ($\phi$) and we conclude that Tweety can fly ($\psi$). However, we are further informed that Tweety is a penguin ($\chi$) so now we must override our original conclusion and finally conclude that Tweety cannot fly ($\neg\psi$).

In Example 3.4, the axiom of monotonicity does not hold because the additional information that Tweety is a penguin ($\chi$) causes us to withdraw our original conclusion ($\psi$).

As far as NMR is concerned, then, the fact that the axiom of monotonicity fails to hold makes its definition a fairly negative one. In [KLM90], the authors set about viewing this negative property in a positive light by proposing various properties which one would expect to hold in nonmonotonic logic.

Now we can define nonmonotonic inference in terms of preferential models.

Definition 3.15 (Nonmonotonic Inference)

$$\phi \vdash_\prec \psi \iff \text{Min}_< (\text{Mod}(\phi)) \subseteq \text{Mod}(\psi)$$

This states that $\psi$ may be nonmonotonically inferred from $\phi$ if and only if the minimal models of $\phi$ (under some arbitrary ordering $<$) are a subset of the models of $\psi$. Figure 3.4 illustrates this.

When this preferential models approach was first developed in this context, the $\prec$-ordering was assumed to be irreflexive, asymmetric and transitive, which makes it different from the $\leq$-ordering defined in Section 3.2.2. The latter is a preorder, i.e. it is reflexive and transitive.

3.4.2 Logical Properties

In this section, eight axioms and inference rules are briefly presented. They are the “positive” properties of NMR as discussed in [Mak94, KLM90]. In fact, this section is largely based on section 3 of [KLM90]. Henceforth, it is assumed that we are taking the preferential models approach so that $\phi \vdash_\prec \psi$ is equivalent to the definition of $\phi \vdash_\prec \psi$ above.
Figure 3.4: An illustration of nonmonotonic inference using the preferential models approach: $\phi \vdash^\prec \psi \iff \text{Min}^\prec(\text{Mod}(\phi)) \subseteq \text{Mod}(\psi)$

1. **Reflexivity**

   $$\alpha \vdash^\alpha$$

   This axiom is the most trivial of all and appears to be satisfied by almost all systems.

2. **Left Logical Equivalence**

   $$\Gamma \models \alpha \iff \beta \quad \alpha \vdash^\gamma \beta \vdash^\gamma$$

   This rule represents the fact that logically equivalent formulae have exactly the same consequences.

3. **Right Weakening**

   $$\Gamma \models \alpha \Rightarrow \beta \quad \gamma \vdash^\alpha \beta \vdash^\gamma$$

   The rule of Right Weakening, as its name suggests, states that we may substitute a weaker formula to the right of the $\vdash$ sign.
4. Cut

\[
\alpha \land \beta \vdash \gamma \quad \alpha \vdash \beta \\
\alpha \vdash \gamma
\]

The Cut rule states that, in order to reach a conclusion \( \gamma \) from a set of facts \( \alpha \), we may first add a new fact \( \beta \) to \( \alpha \) and show that \( \gamma \) may be concluded from this extended set of facts, and then show that \( \beta \) is itself a logical consequence of \( \alpha \).

5. Cautious Monotonicity

\[
\alpha \vdash \beta \quad \alpha \vdash \gamma \\
\alpha \land \beta \vdash \gamma
\]

To recap, monotonicity may be given as the following:

\[
\alpha \vdash \gamma \\
\alpha \land \beta \vdash \gamma
\]

It states that, if we can draw a set of conclusions \( \gamma \) from a set of premises \( \alpha \), then whatever new facts \( \beta \) we add to our set of premises, we will still be able to draw the same set of conclusions.

The rule of Cautious Monotonicity is simply a weaker form of the monotonicity rule. It shows that, as long as the new fact \( \beta \) was already a plausible consequence of the original premises \( \alpha \), then conjunction of \( \beta \) with \( \alpha \) as a premise will not affect the set of conclusions \( \gamma \) which we were able to draw from \( \alpha \) originally.

In terms of preferential models, a special condition must be imposed on the set of models for the property of cautious monotonicity to hold. An example from [Mak94] may be used to illustrate this. Let \( M = \{ m_i \mid i < \infty \} \) and put \( m_j \leq m_i \) iff \( j > i \). Let \( a, b, c \) be three atomic propositions and put \( m_i \models a \) for all \( i \), \( m_i \models b \) for no \( i \), and \( m_i \models c \) for only \( i = 1 \). Then vacuously \( a \vdash b \) and \( a \vdash c \) since \( \text{Min}_<\text{Mod}(a) = \emptyset \), but \( a \land c \not\vdash b \) since \( m_1 \in \text{Min}_<\text{Mod}(a \land c) \) and \( m_1 \not\in \text{Mod}(b) \). Therefore, we stipulate that \( M \) must be stopped (or smooth in [KLM90]). We formally define the property of stoppedness in definition 3.16.

**Definition 3.16 (Stoppedness)** Let \( M \) be an ordered set of models and \( A \) a formula. \( M \) is stopped iff:

\[
\forall m \in M . \, m \in \text{Mod}(A) \rightarrow \exists n \in \text{Min}_<(\text{Mod}(A)) . \, n \leq m
\]
6. **Equivalence**

\[
\alpha \vdash \beta \quad \beta \vdash \alpha \quad \alpha \vdash \gamma \\
\beta \vdash \gamma
\]

This rule states that if two formulae \( \alpha \) and \( \beta \) are logical consequences of each other, then they themselves have the same set of logical consequences \( \gamma \).

7. **And**

\[
\alpha \vdash \beta \quad \alpha \vdash \gamma \\
\alpha \vdash \beta \land \gamma
\]

The And rule expresses the fact that the conjunction of two logical consequences, \( \beta \) and \( \gamma \), of a premise \( \alpha \) is also a plausible consequence of \( \alpha \).

8. **MPC**

\[
\alpha \vdash \beta \Rightarrow \gamma \quad \alpha \vdash \beta \\
\alpha \vdash \gamma
\]

The name of this rule stands for Modus Ponens in the Consequent, and this is exactly what we do.

Note that the last three rules are derivable from (1) to (5) but it is useful to give them explicitly here. Proof of their derivability may be found in [KLM90].

In [KLM90], Kraus, Lehmann and Magidor present many more inference rules which are applicable to certain systems of nonmonotonic logic which have been proposed in the literature. The eight shown above though are those which we would usually expect to be true of any nonmonotonic system. To quote the authors: “It embodies what we think ... are the rockbottom properties without which a system should not be considered a logical system.”

### 3.4.3 Nonmonotonic Reasoning and Belief Revision

In [MG91], the relationship between belief revision and nonmonotonic reasoning is formalised, as Makinson and Gärdenfors provide a formal translation procedure to enable translation of postulates and conditions from one domain to the other.
With the translation procedure in place, Makinson and Gärdenfors go on to translate first the belief revision rationality postulates (see section 3.1.2) and then the nonmonotonic inference conditions found in [Mak94]. The revision postulates all translate into a condition on $\vdash$ that is valid in some kinds of nonmonotonic inference. In particular, translations of all of (K*1)-(K*7) except consistency preservation hold in all classical stoppered preferential model structures. Conversely, each condition on $\vdash$ translates into a condition on * which is derivable from the Gärdenfors postulates without exception.

In some sense, this result justifies our claim that we are drawing upon the nonmonotonic nature of feature integration in this research. In Chapters 4 and 5, we go on to prove that there is a strong correlation between the operations of feature integration and update. Now, in view of what we know about the relationship between update and belief revision (see Sections 3.1.3 and 3.2.3), and the strong relationship between nonmonotonic reasoning and belief revision which we have summarised above, we have shown, albeit indirectly, that there is indeed a relationship between nonmonotonic reasoning and feature integration.

In this chapter, we have reviewed the logic of theory change, in particular, the operations of belief revision and update, and nonmonotonic reasoning. In the following chapters we will show how this theory can be applied to features.
Chapter 4

Propositional Logic Denotation of SMV

In order to reason about SMV feature integration as an update operation, we need to denote SMV programs and features in propositional logic. This will enable us to consider feature integration in terms of Katsuno and Mendelzon’s eight rationality postulates for update, as presented in Chapter 3.

In this chapter, we begin by introducing an abstract representation of SMV programs and features in Section 4.1. This serves as an intermediate step on the way to the propositional logic formulation. Before presenting this though, in Section 4.2 we give details of a denotation which is more natural than that which we finally used, but which is entirely unsuitable for this work, and we explain why. The correct denotation appears in Section 4.3. This is followed by a theorem and proof in Section 4.4 which shows preliminary evidence of a correlation between update and feature integration. Then in Section 4.5, we prove that an equivalent denotation exists to that given in Section 4.3. Finally in Section 4.6, we briefly consider Katsuno and Mendelzon’s rationality postulates for update in terms of SMV programs and features.

The results in this chapter were first published in [HR02].
4.1 An Abstract Representation

The abstract representation of SMV and the feature construct given here results in simplification of the SMV syntax whilst maintaining expressivity. Consequently, it is relatively straightforward to translate from this representation into propositional logic. This intermediate step should also aid understanding of how the denotation works.

Note that we have not attempted to include SMV’s initialisation of values in this abstract representation. At this stage, we want to keep the translation procedure as simple as possible. However, it should be fairly straightforward to extend the representation to incorporate initialisation of variables in future.

**Definition 4.1 (Abstract Representation for an SMV Program)** The abstract representation for an SMV program consists of a tuple $P$:

$$P = \{x_1 : \{\phi_{i1} \rightarrow A_{i1}, \ldots, \phi_{i r_i} \rightarrow A_{i r_i}\}, \ldots, x_n : \{\phi_{n1} \rightarrow A_{n1}, \ldots, \phi_{nr_n} \rightarrow A_{nr_n}\}\}$$

such that:

- $X_P = \{x_1, \ldots, x_n\}$ is the set of variables occurring in $P$.

- The $\phi_{ij}$ are logical formulae over the variables of $P$; they represent the cases to evaluate the next value of $x_i$.

- The $\phi_{ij}$ for a particular variable $x_i$ are covering (that is, they cover every possible state of a transition system) and exclusive (that is, they do not overlap with each other). In other words, for all $x_i$, there is one and only one $\phi_{ij}$ which is true in any given state.

- Each $A_{ij}$ represents a set of possible next values for $x_i$ for case $\phi_{ij}$.

Recall that an SMV system description evaluates each variable’s value in the next state in terms of a set of cases which are based upon the values of variables in the current state. In the representation, then, a set of case-value pairs is associated with each variable $x_i$ which represents the case statements to evaluate the next value of $x_i$ in terms of the current values of variables. In Definition 4.1 above, the $\phi_{ij}$ represent the cases, which
are covering and exclusive for each variable. For example, the intruder deterrent light in Example 2.4 has two case-value pairs associated with the switch variable, the cases for which are switch = on and (by default) switch = off respectively. Each $A_{ij}$ represents a set of possible next values for $x_i$. In other words, each one of these pairs states that ‘if $\phi_{ij}$ is true of the current state then the possible values of variable $x_i$ in the next state are in $A_{ij}$.’ We know that one and only one $\phi_{ij}$ will apply for any given current state because of the covering and exclusive requirement.

It is relatively straightforward to convert an SMV program into its corresponding abstract representation because the case statements translate smoothly into case-value pairs. The only slight complication is that SMV evaluates the cases from top to bottom until it reaches an applicable one and this makes the cases exclusive automatically. We have to conjoin each $\phi_{ij}$ with a negation of all the previous $\phi_{ij}$ in order to ensure exclusivity.

**Example 4.1** The abstract representation for the intruder deterrent light program, IDL, of Example 2.4:

$$\{\text{switch} : \{(\text{switch} = \text{on}) \rightarrow \{\text{off}\}, \neg(\text{switch} = \text{on}) \rightarrow \{\text{on}, \text{off}\}\}\}$$

In this abstract representation, observe that:

- $X_{IDL} = \{\text{switch}\}$ and so $x_1 = \text{switch}$.
- $\phi_{11} = (\text{switch} = \text{on})$ and $\phi_{12} = (\neg(\text{switch} = \text{on}))$.
- $A_{11} = \{\text{off}\}$ and $A_{12} = \{\text{on}, \text{off}\}$.

Note how exclusivity is achieved in Example 4.1. As mentioned before, the reason it is necessary is that, in SMV, case statements are evaluated top to bottom, and the result is the expression from first branch whose condition evaluates to true. However, for this kind of representation, the cases have to made exclusive explicitly.

**Example 4.2** The abstract representation of the level crossing example, LC, of Example 2.5.

$$\{\text{barrier} : \{\text{train\_coming} \rightarrow \{\text{down}\}, \neg\text{train\_coming} \rightarrow \{\text{up}\}\}, \text{train\_coming} : \{1 \rightarrow \{T, F\}\}\}$$

In this abstract representation, observe that:
\[ X_{LC} = \{ \text{barrier, train\_coming} \} \text{ and so } x_1 = \text{barrier and } x_2 = \text{train\_coming}. \]

\[ \phi_{11} = \text{train\_coming} \text{ and } \phi_{12} = \neg \text{train\_coming}. \phi_{21} = 1. \]

\[ \text{A}_1 = \{ \text{down} \} \text{ and } \text{A}_2 = \{ \text{up} \}. \text{A}_2 = \{ T, F \}. \]

A similar approach can be taken to a feature \( F \). It will also have a set of case-value pairs associated with each variable. The only difference is that the cases will not necessarily be covering for a feature because it defers to the program for the cases which are not covered. To help readability, we use \( \alpha, B \) in features where we used \( \phi, A \) in Definition 4.1.

**Definition 4.2 (Abstract Representation for a Feature)** The abstract representation for an SFI feature consists of a tuple \( F \):

\[ F = \{ x_1 : \{ \alpha_{11} \rightarrow B_{11}, \ldots, \alpha_{im_1} \rightarrow B_{1m_1} \}, \ldots, x_i : \{ \alpha_{i1} \rightarrow B_{i1}, \ldots, \alpha_{im_i} \rightarrow B_{im_i} \} \} \]

such that:

\[ \text{X}_F = \{ x_1, \ldots, x_i \} \text{ is the set of variables occurring in } F. \]

\[ \text{The } \alpha_{ik} \text{ are logical formulae over the variables of } F; \text{ they represent the cases to evaluate the next value of } x_i. \]

\[ \text{The } \alpha_{ik} \text{ for a particular variable } x_i \text{ are exclusive (that is, they do not overlap with each other).} \]

\[ \text{Each } B_{ik} \text{ represents a set of possible next values for } x_i \text{ for case } \alpha_{ik}. \]

**Example 4.3** The abstract representation for the intruder deterrent light \texttt{evening\_hours} feature, \( \text{EH} \), of Example 2.6:

\[ \{ \text{switch} : \{ \text{evening\_hours} \rightarrow \{ \text{on} \} \} \} \]

*In this abstract representation, observe that:

\[ \text{X}_{\text{EH}} = \{ \text{switch} \} \text{ and so } x_1 = \text{switch}. \]
• \( \alpha_{11} = \text{evening hours.} \)

• \( B_{11} = \{ \text{on} \} \).

**Example 4.4** The abstract representation for the level crossing's emergency feature, \( EM \), of Example 2.8:

\[
\{ \text{barrier} : \{ \text{emergency} \rightarrow \{ \text{down} \} \} \}
\]

In this abstract representation, observe that:

• \( X_{EM} = \{ \text{barrier} \} \) and so \( x_1 = \text{barrier} \).

• \( \alpha_{11} = \text{emergency} \).

• \( B_{11} = \{ \text{down} \} \).

In Definition 4.3, we show the abstract representation for \( P + F \), that is, the program which results when a feature \( F \) is integrated into a program \( P \). This formulates feature integration as defined in [PR00] and summarised in Section 2.4.2 of this thesis. Note that \( P + F \) is a program in itself so its abstract representation can be obtained from Definition 4.1. Also note that we assume, without loss of generality, that \( X_F \subset X_P \); if a feature refers to a variable whose behaviour is not explicitly specified by a related program, the variable’s behaviour is assumed to be non-deterministic in the program. For example, this is the situation for the evening hours variable in Example 4.3.

**Definition 4.3 (Feature Integration for Abstract Programs)** Let \( P \) and \( F \) be defined as in Definitions 4.1 and 4.2 respectively. Since \( X_F \cap X_P = \emptyset \), for all \( x \), either \( x \in X_F \cap X_P \) or \( x \in X_P \setminus X_F \).

Suppose \( x_i \in X_F \cap X_P \). Then the abstract program \( P + F \) is given by:

\[
(x_i : \{ \alpha_{ik} \rightarrow B_{ik} \mid 1 \leq k \leq m_i \} \\
\cup \{ \neg \alpha_i \land \phi_{ij} \rightarrow A_{ij} \mid 1 \leq j \leq r_i \}) \in P + F
\]

where \( \alpha_i = \bigvee_{j=1}^{m_i} \alpha_{ij} \).

Suppose \( x_i \in X_P \setminus X_F \). Then:

\[
(x_i : \{ \phi_{ij} \rightarrow A_{ij} \mid 1 \leq j \leq r_i \}) \in P + F
\]
In Definition 4.3, the variables are split into two sets: those whose behaviour is specified by the feature and those whose behaviour is not specified by the feature. For the former set of variables, the case-value pairs are taken from the feature and then for the cases which are not covered by the feature, the case-value pairs are taken from the program. For the rest of the variables, the case-value pairs are taken from the program.

**Example 4.5** The abstract representation for the featured program of Example 2.7.

\[
\begin{align*}
&\{\text{switch} : \{\text{evening hours} \to \{on\},} \\
&\quad \neg \text{evening hours} \land (\text{switch} = \text{on}) \to \{\text{off}\},} \\
&\quad \neg \text{evening hours} \land \neg (\text{switch} = \text{on}) \to \{\text{on, off}\}\},} \\
&\text{evening hours} : \{1 \to \{T,F\}\}\}\}
\end{align*}
\]

In this abstract representation, observe that:

- \(X_{\text{IDL}+\text{EH}} = \{\text{switch, evening hours}\}\) and so \(x_1 = \text{switch}\) and \(x_2 = \text{evening hours}\).

- Moreover, \(x_1 \in X_{\text{EH}} \cap X_{\text{IDL}}\) and \(x_2 \in X_{\text{IDL}} \setminus X_{\text{EH}}\).

- \(\neg \alpha_1 = \neg \text{evening hours}\).

In Example 4.5, note how the cases are made exclusive by using negation, and how the behaviour of the \text{evening hours} variable is set to be non-deterministic.

**Example 4.6** The abstract representation for the featured program of Example 2.9.

\[
\begin{align*}
&\{\text{barrier} : \{\text{emergency} \to \{\text{down}\},} \\
&\quad \neg \text{emergency} \land \text{train\_coming} \to \{\text{down}\},} \\
&\quad \neg \text{emergency} \land \neg \text{train\_coming} \to \{\text{up}\}\},} \\
&\text{train\_coming} : \{1 \to \{T,F\}\},} \\
&\text{emergency} : \{1 \to \{T,F\}\}\}\}
\end{align*}
\]

In this abstract representation, observe that:

- \(X_{\text{LC}+\text{EM}} = \{\text{barrier, train\_coming, emergency}\}\) and so \(x_1 = \text{barrier}\), \(x_2 = \text{train\_coming}\) and \(x_3 = \text{emergency}\).
• Moreover, \( x_1 \in X_{EH} \cap X_{IDL}, x_2 \in X_{IDL} \setminus X_{EH} \) and \( x_3 \in X_{IDL} \setminus X_{EH} \).

• \( \neg \alpha_1 = \neg \text{emergency} \).

4.2 A Natural Denotation

In this section, we present the first propositional logic denotation of SMV that we constructed. It is much more obvious and intuitive than the one we finally used in this work; it involves fewer atomic propositions and, in general, is simpler to manage. However, in the context of programs, features and update, it turned out to be entirely unsuitable. We explain this situation here because it is important to understand why such an intuitive formulation is unsuitable. The final denotation is then presented in Section 4.3.

For the propositional logic denotation of a program \( P \), we have a finite set \( \text{Vars}_P = X_P \uplus X'_P \) of \textit{atomic propositions} where, in terms of SMV programs, \( X_P \) is a set of propositions representing the current values of variables and \( X'_P \) is a set of primed propositions representing the possible next values of variables. If the context is clear and the program with which these sets are associated is obvious, then the \( P \) subscripts may be omitted.

\[
X_P = \{ x_i \mid 1 \leq i \leq n \text{ where } n \text{ is the number of variables in } P \}
\]

\[
X'_P = \{ x' \mid x \in X_P \}
\]

So, for each variable, there is one unprimed proposition representing the variable’s current value, and one primed atomic proposition representing its next value.

Note that although we call it \( X_P \), the contents of this set of propositions is different from that of the set \( X_P \) to which we referred in Definition 4.1. Ostensibly, the cardinality and the name of each element of the set is still the same. However, there is a crucial difference. Before, \( X_P \) simply contained the names of all the variables in a program \( P \). Now, it contains a set of atomic propositions, each of which represents a variable in program \( P \).

In this chapter, \( \llbracket P \rrbracket \) denotes the propositional logic denotation of the program \( P \). Likewise for a feature \( F \). Formulae are given by the usual propositional logic grammar - see Definition 3.1.
Definition 4.4 (A Natural Denotation of an SMV Program, $P$) $P$ is defined as in Definition 4.1.

$$[[P]] = \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{r_i} (\phi_{ij} \rightarrow \bigvee_{v \in A_{ij}} (x'_i = v))$$

In this denotation, then, there is a conjunction of implications associated with each variable $x_i$. The antecedents of the implications correspond to the cases $\phi_{ij}$ in terms of the current values of variables, and the consequents consist of a disjunction of the possible next values for $x_i$ in the particular case.

Example 4.7 The denotation of the intruder deterrent light program whose abstract representation is given in Example 4.1.

$$((\text{switch} = \text{on}) \rightarrow (\text{switch}' = \text{off})) \land \neg((\text{switch} = \text{on}) \rightarrow (\text{switch}' = \text{on}) \lor (\text{switch}' = \text{off}))$$

Example 4.8 The denotation of the level crossing program whose abstract representation is given in Example 4.2.

$$(\text{train\_coming} \rightarrow (\text{barrier}' = \text{down})) \land$$

$$(\neg\text{train\_coming} \rightarrow (\text{barrier}' = \text{up})) \land$$

$$(\text{train\_coming}' \lor \neg\text{train\_coming}')$$

Definition 4.5 (A Natural Denotation of an SFI Feature, $F$) $F$ is defined as in Definition 4.2.

$$[[F]] = \bigwedge_{i=1}^{l} \bigwedge_{k=1}^{m_i} (\alpha_{ik} \rightarrow \bigvee_{v \in B_{ik}} (x'_i = v))$$

Example 4.9 The denotation of the feature for the intruder deterrent light whose abstract representation is given in Example 4.3.

$$\text{evening\_hours} \rightarrow \text{switch}' = \text{on}$$

Example 4.10 The denotation of the feature for the level crossing whose abstract representation is given in Example 4.4.

$$\text{emergency} \rightarrow \text{barrier}' = \text{down}$$
Taking a disjunction of the set of next values in this way seems to be a very natural approach but the next example will demonstrate why it fails.

Example 4.11 Consider a simple program $Q$ and a feature $G$, both of which contain a single variable, $y$. The abstract representations of $Q$ and $G$, with their propositional logic denotations, $[Q]$ and $[G]$ respectively, are as follows:

$$Q = \{ y : \{1 \to \{a, b\}\} \}$$
$$[Q] = (1 \to y' = a \lor y' = b)$$
$$G = \{ y : \{1 \to \{a, b, c\}\} \}$$
$$[G] = (1 \to y' = a \lor y' = b \lor y' = c)$$

So the next value of $y$ is always in $\{a, b\}$ in the program. The feature simply changes this set of possible values to $\{a, b, c\}$.

Now we construct $Q + G$ and its denotation, $[Q + G]$:  

$$Q + G = \{ y : \{1 \to \{a, b, c\}, \neg1 \land 1 \to \{a, b\}\} \}$$

$$[Q + G] = (1 \to y' = a \lor y' = b \lor y' = c) \land (\neg1 \land 1 \to y' = a \lor y' = b)$$

Note that the second conjunct of $[Q + G]$ in Example 4.11 will never apply because its antecedent is always false. Conversely, the first conjunct will always apply because its antecedent is always true.

From Example 4.11, it is clear that $[Q] \to [G]$, and this is indicative of a problem with this denotation. Intuitively, if a program implies a feature, it should be as though the feature has already been integrated into the program, just as, in update, if a knowledge base $\psi$ implies a piece of information $\mu$, then effectively, $\psi \circ \mu$ is already achieved. This reasoning is formalised in the axiom (U2) (see Chapter 3). (U2) states that, in this case, $[Q]$ and $[Q + G]$ should be equivalent. However, in this case they are not, so (U2) is not satisfied.

The problem here is that the level of non-determinism in a program corresponds to the strength of the formula which denotes it. Usually we expect the strength of a formula to correspond to the amount of information which it contains. In Example 4.11, $G$ is more non-deterministic than $Q$ but they contain the same amount of information about
the possible next values of $y$. That is, they both specify exactly which values $y$ is allowed to take in the next state, and which values it is not allowed to take. Furthermore, we would not wish $Q$ to imply $G$ because $G$ specifies different behaviour from $Q$ and should therefore alter the behaviour of $Q$.

The solution is to find a denotation which renders $Q$ and $G$ equally strong, in the sense that they are incomparable and there is no implication in either direction. In the final denotation, we make the temporal nature of these formulae more explicit. We strengthen the representation of a variable’s possible next values from a disjunction to a conjunction of both the possible and the impossible next values. This will be presented formally in Section 4.3. A similar problem to that illustrated by Example 4.11 arose when we simply changed the disjunction to a conjunction of possible next values without considering the next values which were not allowed. In fact, the final denotation means that if one program implies another, they will be equivalent. For the moment, we find this strong assertion to be satisfactory in the context of SMV programs in view of the fact that the so-called impossible values for a variable are implicit in terms of the values which a program specifies to be possible.

\section{The Correct Denotation}

$X_P$ is defined as before but now $X'_P$ consists of several subsets, one for each variable $x_i \in X_P$.

$$X_P = \{ x_i \mid 1 \leq i \leq n \text{ where } n \text{ is the number of variables in } P \}$$

$$X'_P = \bigcup_{x_i \in X_P} \{ x'_{i,v} \mid v \in \text{type}(x_i) \}$$

\text{type}(x) denotes a finite set of possible values for $x$, e.g. if $x$ is boolean, $\text{type}(x) = \{ T,F \}$. $X_P$ contains the propositions denoting the current values of variables. There is a subset of $X'_P$ for each variable $x \in X_P$ which contains propositions relating to the possible next values for $x$ according to its type. The intended literal meaning of the proposition $x'_{i,v}$ is ‘$v$ is a possible value for variable $x_i$ in the next state.’ Thus, this new set of propositions encodes the temporal aspect of SMV explicitly.
Propositional Logic Denotation of SMV

Example 4.12 Consider the intruder deterrent light program (see its abstract representation in Example 4.1). The two sets of propositions for IDL are as follows:

\[ X_{IDL} = \{ \text{switch} \} \]
\[ X'_{IDL} = \{ \text{switch}^i_{on}, \text{switch}^i_{off} \} \]

Example 4.13 Consider the level crossing program (see its abstract representation in Example 4.2). The two sets of propositions for LC are as follows:

\[ X_{LC} = \{ \text{barrier, train\_coming} \} \]
\[ X'_{LC} = \{ \text{barrier}^i_{up}, \text{barrier}^i_{down}, \text{train\_coming}^i_{T}, \text{train\_coming}^i_{P} \} \]

Now we can consider the revised denotations of \( P, F \) and \( P + F \).

Definition 4.6 (Denotation of an SMV Program, \( P \)) Let \( P \) be defined as in Definition 4.1.

\[
[P] = \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (\phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} x^i_{j, v} \wedge \bigwedge_{v \not\in A_{ij}} \neg x^i_{j, v})
\]

where \( x^i_{j, v} \in X^i_{P} \), and \( v \not\in A_{ij} \) is shorthand for \( v \in \text{type}(x_i) \setminus A_{ij} \).

In this denotation, the case-value pairs for each variable \( x_i \) are still encoded as a conjunction of implications. However, this time the consequents consist of two conjunctions: one for the values which are possible for \( x_i \) in the next state and one for those values which are \textit{not} possible. This gives an explicit representation of a variable’s possible next values.

Example 4.14 The denotation of the abstract program of Example 4.1.

\((\text{switch} = \text{on}) \rightarrow \text{switch}^i_{off} \wedge \neg \text{switch}^i_{on}) \wedge (\neg (\text{switch} = \text{on}) \rightarrow \text{switch}^i_{on} \wedge \text{switch}^i_{off})\)

Note that, for the sake of clarity, we use the notation \( \text{switch} = \text{on} \), rather than using a boolean proposition called \( \text{switch} \). This maintains generality because \( \text{switch} \) is a binary variable. If we were dealing with \( n \)-ary enumerated variables, we could still encode them
using propositions. For example, if a variable had 8 different possible values, then it could be encoded using 3 propositions (because $2^3 = 8$). However, throughout this thesis, we have chosen to use binary variables because we want to keep the systems as simple as possible at this stage.

**Example 4.15**  The denotation of the abstract program of Example 4.2.

\[
(\text{train\textunderscore coming} \rightarrow \text{barrier}^\prime_{\text{down}} \land \neg \text{barrier}^\prime_{\text{up}}) \land \\
(\neg \text{train\textunderscore coming} \rightarrow \text{barrier}^\prime_{\text{up}} \land \neg \text{barrier}^\prime_{\text{down}}) \land \\
(\text{train\textunderscore coming}^\prime_T \land \text{train\textunderscore coming}^\prime_F)
\]

**Definition 4.7 (Denotation of a Feature, $F$)** Let $F$ be defined as in Definition 4.2.

\[
[F] = \bigwedge_{i=1}^{m_i} \left( a_{ik} \rightarrow \bigwedge_{v \in B_{ik}} x^i_{tv} \land \bigwedge_{v \notin B_{ik}} \neg x^i_{tv} \right)
\]

where $x^i_{tv} \in X^t_F$ and $v \notin B_{ik}$ is shorthand for $v \in \text{type}(x_i) \setminus B_{ik}$.

Definition 4.7 is logically identical to Definition 4.6 but we need it in order that we have the syntax to differentiate between $P$ and $F$ throughout the rest of this chapter.

**Example 4.16**  The denotation of the abstract feature of Example 4.3.

\[
e\text{evening} \land \text{hours} \rightarrow \text{switch}^\prime_{\text{on}} \land \neg \text{switch}^\prime_{\text{off}}
\]

**Example 4.17**  The denotation of the abstract feature of Example 4.4.

\[
e\text{emergency} \rightarrow \text{barrier}^\prime_{\text{down}} \land \neg \text{barrier}^\prime_{\text{up}}
\]

**Lemma 4.1 (Denotation of a Featured Program, $P + F$)** Let $P$ and $F$ be defined as in Definitions 4.1 and 4.2 respectively.

Without loss of generality, we assume the variables which are referred to in the feature, that is, the set $x_i \in X_F$, to be $x_1, \ldots, x_i$ and those which are not, that is, the set $x_i \in X_P$. 

...
Propositional Logic Denotation of SMV

$X_P \setminus X_F$, to be $x_{i+1}, \ldots, x_n$.

$$\left[ P + F \right] = \bigwedge_{i=1}^{l} \left( \bigwedge_{k=1}^{m_i} (\alpha_i \land \bigwedge_{v \in B_{ik}} x_{i,v}^l \land \bigwedge_{v \notin B_{ik}} \neg x_{i,v}^l) \land \bigwedge_{j=1}^{r_i} (\neg \alpha_i \land \phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} x_{i,v}^l \land \bigwedge_{v \notin A_{ij}} \neg x_{i,v}^l) \right) \land \bigwedge_{i=l+1}^{n} \left( \bigwedge_{j=1}^{r_i} (\phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} x_{i,v}^l \land \bigwedge_{v \notin A_{ij}} \neg x_{i,v}^l) \right)$$

where $\alpha_i = \bigvee_{k=1}^{m_i} \alpha_{ik}$ and $x_{i,v}^l \in X_{P,F}^l$.

For each variable in the feature, the first line of Lemma 4.1 represents behaviour of the feature and the second line defers to the program for the cases which are not covered by the feature. $\neg \alpha_i$ is true when none of the feature’s $x_i$ cases are true. The third line represents behaviour of variables which are specified entirely by the program, not the feature.

**Example 4.18** The denotation of the abstract featured program of Example 4.5 is the following:

$$(\text{evening hours} \rightarrow \text{switch}^l_{on} \land \neg \text{switch}^l_{off}) \land$$

$$(\neg \text{evening hours} \land (\text{switch} = \text{on}) \rightarrow \text{switch}^l_{off} \land \neg \text{switch}^l_{on}) \land$$

$$(\neg \text{evening hours} \land \neg (\text{switch} = \text{on}) \rightarrow \text{switch}^l_{on} \land \text{switch}^l_{off}) \land$$

$$(\text{evening hours}^l_T \land \text{evening hours}^l_F)$$

**Example 4.19** The denotation of the abstract featured program of Example 4.6 is the following:

$$(\text{emergency} \rightarrow \text{barrier}^l_{down} \land \neg \text{barrier}^l_{up}) \land$$

$$(\neg \text{emergency} \land \text{train\_coming} \rightarrow \text{barrier}^l_{down} \land \neg \text{barrier}^l_{up}) \land$$

$$(\neg \text{emergency} \land \neg \text{train\_coming} \rightarrow \text{barrier}^l_{up} \land \neg \text{barrier}^l_{down}) \land$$

$$(\text{train\_coming}^l_T \land \text{train\_coming}^l_F) \land$$

$$(\text{emergency}^l_T \land \text{emergency}^l_F)$$
In Example 4.11, we showed why our initial, more intuitive denotation was flawed. In Example 4.20, we revisit that example to show how, in the context of the correct denotation, the counter-example is no longer valid.

**Example 4.20** Recall program $Q$ and feature $G$ of Example 4.11. Using the correct denotation, we present $[Q]$, $[G]$, and $[Q + G]$ here:

$$[Q] = 1 \rightarrow y_a \land y_b \land \neg y_c$$

$$[G] = 1 \rightarrow y_d \land y_b \land y_e$$

$$[Q + G] = (1 \rightarrow y_d \land y_b \land y_e) \land (\neg 1 \land 1 \rightarrow y_a \land y_b \land \neg y_c)$$

Using this denotation, neither $[Q] \rightarrow [G]$, nor $[G] \rightarrow [Q]$: the program and the feature are incomparable and neither formula is weaker than the other.

The comparison which can be made is this: $[Q + G] \leftrightarrow [G]$, the feature is equivalent to the featured program. This, of course, satisfies (U1), the axiom of success.

### 4.4 Evidence of a Correlation

In this section, we define the update operation to be feature integration, as formulated above, and prove that the eight rationality postulates, (U1)-(U8), hold for this operation.

First, we give the formal definition of the update operation in this context. This is followed by the theorem and then proofs of (U1)-(U8). Note that certain cases need not be considered in the proofs because we are dealing with the special case where all formulae represent programs and features. This means that some cases simply do not apply. Such cases can be found within the proofs of (U5), (U6), (U7) and (U8).

**Definition 4.8** We define the update operator on formulae which are denotations of programs and features. Let $P$ and $F$ be an SMV program and an SFI feature respectively.

$$\psi \odot \mu \overset{\text{def}}{=} [P + F] \text{ if } \psi = [P] \text{ and } \mu = [F]$$

$\psi \odot \mu$ is undefined otherwise.
Theorem 4.1 Let $\psi, \psi_1, \psi_2$ be denotations of programs, let $\mu, \mu_1, \mu_2, \phi$ be denotations of features, and let $\circ$ be defined as in Definition 4.8 above. Then, provided the subformulae occurring in them are defined, the axioms (U1)-(U8) hold.

Proof of Theorem 4.1. We need to prove that each rationality postulate for update holds in the context of Definition 4.8. We assume that $P = P_1$, $F = F_1$, and that $[P]$, $[F]$, $[P + F]$, $[P_2]$, $[F_2]$ and $[P_2 + F_2]$ take the forms listed below.

$$[P] = [P_1] \overset{\text{def}}{=} \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{r_i} (\phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}}^{x'_{i,v}} \wedge \bigwedge_{v \not\in A_{ij}}^{\neg x'_{i,v}})$$

where $x'_{i,v} \in X'_P$ and $v \not\in A_{ij}$ is shorthand for $v \in \text{type}(x_i) \setminus A_{ij}$.

$$[F] = [F_1] \overset{\text{def}}{=} \bigwedge_{i=1}^{l} \bigwedge_{k=1}^{m_i} (\alpha_{ik} \rightarrow \bigwedge_{v \in B_{ik}}^{x'_{i,v}} \wedge \bigwedge_{v \not\in B_{ik}}^{\neg x'_{i,v}})$$

where $x'_{i,v} \in X'_F$ and $v \not\in B_{ik}$ is shorthand for $v \in \text{type}(x_i) \setminus B_{ik}$.

$$[P + F] = [P_1 + F_1] \overset{\text{def}}{=} \bigwedge_{i=1}^{l} \bigwedge_{k=1}^{m_i} (\alpha_{ik} \rightarrow \bigwedge_{v \in B_{ik}}^{x'_{i,v}} \wedge \bigwedge_{v \not\in B_{ik}}^{\neg x'_{i,v}}) \wedge$$

$$\bigwedge_{j=1}^{r_i} (\neg \alpha_1 \wedge \phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}}^{x'_{i,v}} \wedge \bigwedge_{v \not\in A_{ij}}^{\neg x'_{i,v}}) \wedge$$

$$\bigwedge_{i=l+1}^{n} \bigwedge_{j=1}^{r_i} (\phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}}^{x'_{i,v}} \wedge \bigwedge_{v \not\in A_{ij}}^{\neg x'_{i,v}})$$

where $\alpha_i = \bigvee_{k=1}^{m_i} \alpha_{ik}$ and $x'_{i,v} \in X'_{P + F}$.

$$[P_2] \overset{\text{def}}{=} \bigwedge_{i=1}^{p} \bigwedge_{j=1}^{r_i} (\eta_{ij} \rightarrow \bigwedge_{v \in C_{ij}}^{x'_{i,v}} \wedge \bigwedge_{v \not\in C_{ij}}^{\neg x'_{i,v}})$$

where $x'_{i,v} \in X'_{P_2}$ and $v \not\in C_{ij}$ is shorthand for $v \in \text{type}(x_i) \setminus C_{ij}$.

$$[F_2] \overset{\text{def}}{=} \bigwedge_{i=1}^{q} \bigwedge_{k=1}^{m_i} (\gamma_{ik} \rightarrow \bigwedge_{v \in D_{ik}}^{x'_{i,v}} \wedge \bigwedge_{v \not\in D_{ik}}^{\neg x'_{i,v}})$$
where \( x'_{i,v} \in X'_{F_2} \) and \( v \notin D_{ik} \) is shorthand for \( v \in \text{type}(x_i) \setminus D_{ik} \).

\[
[P_2 + F_2] \overset{\text{def}}{=} \bigwedge_{i=1}^{m_i} \left( \bigwedge_{k=1}^{q_i} (\gamma_{ik} \rightarrow \bigwedge_{v \in D_{ik}} x'_{i,v} \land \bigwedge_{v \notin D_{ik}} -x'_{i,v}) \land \right.
\bigwedge_{j=1}^{r_i} (-\gamma_i \land \eta_{ij} \rightarrow \bigwedge_{v \in C_{ij}} x'_{i,v} \land \bigwedge_{v \notin C_{ij}} -x'_{i,v}) \right) \land \bigwedge_{i=q+1}^{p} \left( \bigwedge_{j=1}^{r_i} (\eta_{ij} \rightarrow \bigwedge_{v \in C_{ij}} x'_{i,v} \land \bigwedge_{v \notin C_{ij}} -x'_{i,v}) \right)
\]

where \( \gamma_i = \bigvee_{k=1}^{m_i} \gamma_{ik} \) and \( x'_{i,v} \in X'_{F_2 + F_2} \).

Throughout this proof we will continue to use \( A_{ij}, B_{ik}, C_{ij} \) and \( D_{ik} \) as shorthand for
\[
\bigwedge_{v \in A_{ij}} x'_{i,v} \land \bigwedge_{v \notin A_{ij}} -x'_{i,v}, \ \bigwedge_{v \in B_{ik}} x'_{i,v} \land \bigwedge_{v \notin B_{ik}} -x'_{i,v}, \ \bigwedge_{v \in C_{ij}} x'_{i,v} \land \bigwedge_{v \notin C_{ij}} -x'_{i,v} \text{ and } \bigwedge_{v \notin D_{ik}} -x'_{i,v} \text{ respectively.}
\]

(U1) \( \psi \circ \mu \) implies \( \mu \).

Let \( \mu = [F] \) and \( \psi \circ \mu = [P + F] \).

(U1) is obviously true because the clauses of \([F]\) are clearly included in the first line of \([P + F]\).

(U2) If \( \psi \) implies \( \mu \) then \( \psi \circ \mu \) is equivalent to \( \psi \).

Let \( \psi = [P] \), \( \mu = [F] \) and \( \psi \circ \mu = [P + F] \).

Assume \([P] \rightarrow [F]\).

The \( \rightarrow \) direction.

Assume \([P + F]\).

Prove \([P]\).

For each variable \( x_i \), there are several cases:

1. \( x_i \in X_F \). There are two further cases:

   (a) \( \alpha_{ik} \rightarrow B_{ik} \) from \( F \) is triggered.

      In this case, we know that, for every \( \alpha_{ik} \rightarrow B_{ik} \) clause for \( x_i \) in \([F]\) there is a \( \phi_{ij} \rightarrow A_{ij} \) clause for \( x_i \) in \([P]\) such that \( \phi_{ij} \rightarrow A_{ij} \) implies \( \alpha_{ik} \rightarrow B_{ik} \)
(because $[P] \rightarrow [F]$). As the $\phi_{ij}$ are covering and exclusive, the clauses of the first line of $[P + F]$ will cover the cases which are not covered by the second line of $[P + F]$ (and only those). For these first-line cases we know that $A_{ij} \rightarrow B_{ik}$ holds, and because $A_{ij}$ and $B_{ik}$ are simply large conjunctions of a fixed set of propositions, $A_{ij} \rightarrow B_{ik}$ iff $A_{ij} \leftrightarrow B_{ik}$. So in the first line of $[P + F]$, the part of $[P]$ which is not covered in the second and third lines of $[P + F]$, is covered.

(b) $\neg \alpha_i$ is true so one and only one $\phi_{ij} \rightarrow A_{ij}$ from $P$ is triggered.

$[P]$ is trivial in this case (from the second line of $[P + F]$).

2. $x_i \in X_P \setminus X_F$.

In this case, $[P]$ is trivial (from the third line of $[P + F]$).

**The $\leftarrow$ direction.**

Assume $[P]$.

Prove $[P + F]$.

Again, the main non-trivial case is (1a) as before, i.e. for $x \in X_F$ when $F$ is triggered. However, this is straightforward because, having assumed $[P] \rightarrow [F]$ and $[P]$, we get $[F]$ and, for this case, it is only the first line of $[P + F]$ which we need to prove.

So we have $[P + F] \leftrightarrow [P]$.

**(U3)** If both $\psi$ and $\mu$ are satisfiable then $\psi \circ \mu$ is also satisfiable.

Let $\psi = [P]$, $\mu = [F]$ and $\psi \circ \mu = [P + F]$.

Assume $[P]$ and $[F]$ are satisfiable.

Prove that $[P + F]$ is also satisfiable.

The first and third lines of $[P + F]$ contain entire clauses from $[F]$ and $[P]$ respectively, so they must be satisfiable.

The second line of $[P + F]$ makes the cases covering by deferring to $P$ whenever $F$ is not triggered. So effectively these are clauses from $[P]$ which we know are
(U4) If $\psi_1 \leftrightarrow \psi_2$ and $\mu_1 \leftrightarrow \mu_2$ then $\psi_1 \odot \mu_1 \leftrightarrow \psi_2 \odot \mu_2$.

Let $\psi_1 = [P_1]$, $\psi_2 = [P_2]$, $\mu_1 = [F_1]$, $\mu_2 = [F_2]$, $\psi_1 \odot \mu_1 = [P_1 + F_1]$ and $\psi_2 \odot \mu_2 = [P_2 + F_2]$.

1. Assume $[P_1] \leftrightarrow [P_2]$.

The $\rightarrow$ direction.

Assume $[P_1 + F_1]$.
Prove $[P_2 + F_2]$.

For each variable $x_i$, there are several cases:

1. $x_i \in X_{F_1}$ (so $x_i \in X_{F_2}$ due to assumption 2).
   (a) One and only one $\alpha_{ij} \rightarrow B_{ij}$ from $F_1$ is triggered. There will be some $\gamma_{ik} \rightarrow D_{ik}$ from $F_2$ which is also triggered (assumption 2) such that $B_{ij} = D_{ik}$.
   (b) $\neg \alpha_i$ is true so one and only one $\phi_{ij} \rightarrow A_{ij}$ from $P_1$ is triggered. There will be some $\tau_{kk} \rightarrow C_{ik}$ from $P_2$ which is also triggered (assumption 1), such that $A_{ij} = C_{ik}$.

2. $x_i \in X_{F_1} \setminus X_{F_2}$ (so $x_i \in X_{P_2} \setminus X_{F_2}$ due to assumptions 1 and 2).
   One and only one $\phi_{ij} \rightarrow A_{ij}$ from $P_1$ is triggered. There will be some $\tau_{kk} \rightarrow C_{ik}$ from $P_2$ which is also triggered (assumption 1), such that $A_{ij} = C_{ik}$.

So $[P_2 + F_2]$ is true because for every action of $[P_1 + F_1]$, there is an equivalent corresponding action of $[P_2 + F_2]$.

So $[P_1 + F_1] \rightarrow [P_2 + F_2]$.

The $\leftarrow$ direction.

We reason likewise to prove $[P_2 + F_2] \rightarrow [P_1 + F_1]$.

So we have $[P_1 + F_1] \leftrightarrow [P_2 + F_2]$. 

Propositional Logic Denotation of SMV
\((U5)\) \((\psi \circ \mu) \land \phi\) implies \(\psi \circ (\mu \land \phi)\).

Let \(\psi = [P], \mu = [F_1], \phi = [F_2], \psi \circ \mu = [P + F_1].\)

The formulation of this postulate is interesting because in the right-hand side of the postulate we need to take \(\mu \land \phi\) (that is \([F_1] \land [F_2]\)) such that the result is the denotation of a valid SFI feature, say \([F']\), in order that we can then take \([P + F']\).

Often, the conjunction of two features, \([F_1] \land [F_2]\), leads to a contradiction and no valid SFI feature will ever be denoted by \(\bot\). For instance, when each feature specifies different behaviour for a variable \(x\) in a particular case, their respective formulae will contradict each other. We have only defined the \(\circ\) operator on programs and features and so, for any SMV program \(P\), \([P] \circ \bot\) is undefined. Therefore, it is impossible to formulate \((U5)\) in this case.

However, there are cases when \([F']\) is a valid SFI feature. They are as follows:

1. \(\llbracket F_1 \rrbracket \rightarrow \llbracket F_2 \rrbracket\). This is the case iff the following conditions hold:
   - \(X_{F_3} \subseteq X_{F_1}\).
   - For each \(x_i \in X_{F_2}\), \(\gamma_i \rightarrow \alpha_i\).
   - For each \(\gamma_{ik}\), there exists \(\alpha_{ij}\) (or \(\alpha_{i1}, \ldots, \alpha_{ir}\)) such that \(\gamma_{ik} \rightarrow \alpha_{ij}\) (or \(\gamma_{ik} \rightarrow \alpha_{i1} \lor \ldots \lor \alpha_{ir}\)) and \(D_{ik} = B_{ij}\) (or \(D_{ik} = B_{i1} = \ldots = B_{ir}\)).

   In this case, \(\llbracket F_1 \rrbracket \land \llbracket F_2 \rrbracket \leftrightarrow \llbracket F_1 \rrbracket\)

2. \(\llbracket F_2 \rrbracket \rightarrow \llbracket F_1 \rrbracket\). The converse conditions to those for \(\llbracket F_1 \rrbracket \rightarrow \llbracket F_2 \rrbracket\) will hold in this case. In this case, \(\llbracket F_1 \rrbracket \land \llbracket F_2 \rrbracket \leftrightarrow \llbracket F_2 \rrbracket\).

3. \(\llbracket F_1 \rrbracket \leftrightarrow \llbracket F_2 \rrbracket\). In this case, the conditions for both the previous two cases hold.

First we assume \((\llbracket P + F_1 \rrbracket) \land \llbracket F_2 \rrbracket\) and then prove \((U5)\) for the three cases outlined above.

For case 1, we need to prove \(\llbracket P + F_1 \rrbracket\). We know (from \((U1)\)) that \(\llbracket P + F_1 \rrbracket \rightarrow \llbracket F_1 \rrbracket\) and that \(\llbracket F_1 \rrbracket \rightarrow \llbracket F_2 \rrbracket\). Therefore, \((\llbracket P + F_1 \rrbracket) \land \llbracket F_2 \rrbracket \leftrightarrow \llbracket P + F_1 \rrbracket\).

For case 2, we need to prove \(\llbracket P + F_2 \rrbracket\). This is vacuously true if \(\llbracket P + F_1 \rrbracket \land \llbracket F_2 \rrbracket \rightarrow \bot\) (which will be the case if \(\llbracket F_2 \rrbracket\) contradicts the \(P\) part of \(\llbracket P + F_1 \rrbracket\)). Otherwise, it must
be the case that $[P + F_1] \rightarrow [F_2]$ and we know that $F_1$ has altered behaviour of $P$
which $F_2$ would have altered (because $[F_2] \rightarrow [F_1]$) and in order for $[P + F_1] \rightarrow [F_2]$ to hold, the remaining behaviour of $F_2$ must be equivalent to that of $P$. Therefore, we have $[P + F_2]$.
For case 3, (U5) is trivial when $[F_1] \leftrightarrow [F_2]$.

(U6) If $\psi \circ \mu_1$ implies $\mu_2$ and $\psi \circ \mu_2$ implies $\mu_1$ then $\psi \circ \mu_1 \leftrightarrow \psi \circ \mu_2$.

Let $\mu_1 = [F_1]$, $\mu_2 = [F_2]$, $\psi \circ \mu_1 = [P + F_1]$ and $\psi \circ \mu_2 = [P + F_2]$.

Assume:


The $\rightarrow$ direction.

Assume:

3. $[P + F_1]$.

Prove $[P + F_2]$.

We know $[F_2]$ from assumptions 1 and 3.

For each variable $x$, there are 4 cases:

(i) $x \in X_P \cap X_{F_1} \cap X_{F_2}$.
(ii) $x \in X_P \cap X_{F_1} \setminus X_{F_2}$.
(iii) $x \in X_P \cap X_{F_2} \setminus X_{F_1}$.
(iv) $x \in X_P \setminus (X_{F_1} \cup X_{F_2})$.

For case (iv), $x \notin X_{F_1}$ and $x \notin X_{F_2}$ so the set of implications for $x$ in $P$ remains unaltered in both $[P + F_1]$ and $[P + F_2]$.

Case (ii) is impossible if assumption 2 holds unless the $x$-part of $[F_1]$ is equivalent to the $x$-part of $[P]$, in which case the $x$-part of $[P + F_1]$ is equivalent to the $x$-part of $[P]$. 

Case (iii) is impossible if assumption 1 holds unless the $x$-part of $[F_2]$ is equivalent to the $x$-part of $[P]$, in which case the $x$-part of $[P + F_2]$ is equivalent to the $x$-part of $[P]$. So we have $[P + F_2]$ in this case.

Case (i) splits into 4 further cases:

(a) The case is covered by both features.

(b) The case is covered by $F_1$, not $F_2$.

(c) The case is covered by $F_2$, not $F_1$.

(d) The case is not covered by either feature.

For case (d), the program will always be triggered.

If case (b) existed, $F_1$ would have to simply repeat what $P$ did in order that assumption 2 held. If this is so, then we have $[P + F_2]$ in this case.

In case (c), assumption 1 could not hold unless $F_2$ repeated what $P$ did. If so, then, again, we have $[P + F_2]$.

Case (a) concerns cases covered by both features. The fact that we know $[F_2]$ holds means that $F_1$ must behave exactly as $F_2$ does for cases covered by both features. So we have $[P + F_2]$.

For every possible case, we have $[P + F_2]$.

Therefore $[P + F_1] \rightarrow [P + F_2]$.

**The $\leftarrow$ direction.**

We reason symmetrically to get $[P + F_2] \rightarrow [P + F_1]$.

Therefore $[P + F_1] \leftrightarrow [P + F_2]$.

**U7** If $\psi$ is complete then $(\psi \circ \mu_1) \land (\psi \circ \mu_2)$ implies $\psi \circ (\mu_1 \lor \mu_2)$.

Let $\psi = [P]$. By definition, $[P]$ will not be complete so this postulate does not apply in this context.
\[(U8) (\psi_1 \lor \psi_2) \circ \mu \leftrightarrow (\psi_1 \circ \mu) \lor (\psi_2 \circ \mu).\]

Let \(\psi_1 = [P_1], \psi_2 = [P_2]\) and \(\mu = [F]\). We require that the result of \([P_1] \lor [P_2]\) is the denotation of a program, say \([P']\), in order that we can then take \([P' + F]\). A disjunction of two programs does not usually yield a valid SMV program; it will be more like a program specification because it will specify two possible sets of next states (which are likely to be different) for each case. Therefore, the \(\circ\) operation is not defined for this case.

However, in the case where \([P_1] \leftrightarrow [P_2], [P_1] \lor [P_2]\) will indeed yield a program. \((U8)\) will clearly hold in this case.

### 4.5 An Equivalent Denotation

We have found equivalent denotations of \([P]\) and \([P + F]\). They are both based upon disjunctions of the possible models of each formula. The reason why these alternative formulations are relevant to this work is that there is a model-theoretic definition of the update operation and we will need to make use of these equivalent formulations when we prove that feature integration is an update operator in Chapter 5. These equivalences are given below in Theorems 4.2 and 4.3 respectively.

**Notation**

1. \(\{a_1, \ldots, a_{2^n}\}\) represents all the valuations of \(X \subseteq \text{Vars}\).
2. \(\psi_a\) is the conjunctive formula describing the assignment \(a\).
3. Given an assignment \(a\) and a variable \(x_i, j(i,a)\) is defined to be the unique \(j\) such that \(a \models \phi_{ij}\).

**Theorem 4.2** For the SMV program \(P\), the abstract representation of which was given in Definition 4.1:

\[
\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{r_i} (\phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} x_{i,v}^j \land \bigwedge_{v \notin A_{ij}} -x_{i,v}^j) \Leftrightarrow \bigvee_{y=1}^{2^n} (\psi_{a_y} \land \bigwedge_{i=1}^{n} (\bigwedge_{v \in A_{ij} \cap (i,a_y)} x_{i,v}^j \land \bigwedge_{v \notin A_{ij} \cup (i,a_y)} -x_{i,v}^j)) \quad (4.1)
\]

where \(x_{i,v}^j \in X^j_P\).
Proof of Theorem 4.2.

We refer to the original and the new denotations in equation 4.1 as LHS and RHS respectively.

For each assignment $a_y$, and for each variable $x_i$, there is a $\phi_{ij}$ such that $a_y \rightarrow \phi_{ij}$ (because we know that the $\phi_{ij}$ are covering and exclusive).

Moreover, for each $\phi_{ij}$, there is a set of conjunctive formulae, $\psi_{a_1}, \ldots, \psi_{a_w}$ such that $\phi_{ij} \leftrightarrow \psi_{a_1} \lor \ldots \lor \psi_{a_w}$. We can also refer to this set of ‘$\psi$-formulae’ as: $\{ \psi_a | a \models \phi_{ij} \}$.

In order to prove that $LHS \Leftrightarrow RHS$, we will show that, for any $a$, $\psi_a \land LHS \Leftrightarrow \psi_a \land RHS$.

In fact, it is sufficient to show that, for any $a$, $\psi_a \land LHS_i \Leftrightarrow \psi_a \land RHS_i$ where, for some $i$:

$$LHS_i \overset{\text{def}}{=} \bigwedge_{j=1}^{r_i} (\phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} \psi'_{i,v} \land \bigwedge_{v \notin A_{ij}} \neg \psi'_{i,v})$$

$$RHS_i \overset{\text{def}}{=} \bigvee_{y=1}^{2^n} (\psi_{a_y} \land (\bigwedge_{v \in A_{ij}(i,y)} \psi'_{i,v} \land \bigwedge_{v \notin A_{ij}(i,y)} \neg \psi'_{i,v}))$$

Take an assignment, $a_y$.

$$\psi_{a_y} \land LHS_i = \psi_{a_y} \land \bigwedge_{j=1}^{r_i} (\phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} \psi'_{i,v} \land \bigwedge_{v \notin A_{ij}} \neg \psi'_{i,v})$$

Since $a_y$ makes each $\phi_{ij}$ false except one, namely $\phi_{ij[i,a]}$:

$$\psi_{a_y} \land LHS_i = \psi_{a_y} \land (\bigwedge_{v \in A_{ij}(i,a)} \psi'_{i,v} \land \bigwedge_{v \notin A_{ij}(i,a)} \neg \psi'_{i,v})$$

$$\psi_{a_y} \land RHS_i = \psi_{a_y} \land (\bigwedge_{v \in A_{ij}(i,a)} \psi'_{i,v} \land \bigwedge_{v \notin A_{ij}(i,a)} \neg \psi'_{i,v})$$

Therefore, $\psi_{a_y} \land LHS_i \Leftrightarrow \psi_{a_y} \land RHS_i$. 


Theorem 4.3 For the featured program $P + F$, the abstract representation of which is given in Definition 4.3:

$$
\bigwedge_{i=1}^{l} \left( \bigwedge_{k=1}^{m_i} \left( \alpha_{ik} \rightarrow \bigwedge_{v \in B_{ik}} x'_{i,v} \land \bigwedge_{v \notin B_{ik}} \neg x'_{i,v} \right) \land \\
\bigwedge_{j=1}^{r_i} \left( \neg \alpha_i \land \phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} x'_{i,v} \land \bigwedge_{v \notin A_{ij}} \neg x'_{i,v} \right) \land \\
\bigwedge_{i=t+1}^{n} \left( \bigwedge_{j=1}^{r_i} \left( \phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} x'_{i,v} \land \bigwedge_{v \notin A_{ij}} \neg x'_{i,v} \right) \right) \right) \\
\iff \\
\bigvee_{y=1}^{2^n} \left( \psi_{a_y} \land \left( \bigwedge_{i=1}^{l} \left( \bigvee_{k=1}^{m_i} \left( \alpha_{ik} \land \bigwedge_{v \in B_{ik}} x'_{i,v} \land \bigwedge_{v \notin B_{ik}} \neg x'_{i,v} \right) \lor \\
\left( \neg \alpha_i \land \bigwedge_{v \in A_{ij}(i,y)} x'_{i,v} \land \bigwedge_{v \notin A_{ij}(i,y)} \neg x'_{i,v} \right) \right) \land \\
\left( \bigwedge_{i=t+1}^{n} \left( \bigwedge_{j=1}^{r_i} \left( x'_{i,v} \land \bigwedge_{v \notin A_{ij}(i,y)} \neg x'_{i,v} \right) \right) \right) \right) \\
\right) \\
$$

where $\alpha_i = \bigvee_{k=1}^{m_i} \alpha_{ik}$ and $x'_{i,v} \in X'_{P + F}$.

Proof of Theorem 4.3.

We refer to the original and the new denotations in Theorem 4.3 as LHS and RHS respectively.

Let $\{a_1, \ldots, a_{2^n}\}$ be all the valuations of $X \subseteq \text{Vars}$.

Let $\psi_a$ be the conjunctive formula describing the assignment $a$.

For each $a_y$, and for each variable $x_i$, there is a $\phi_{ij}$ such that $a_y \rightarrow \phi_{ij}$ (because we know that the $\phi_{ij}$ are covering and exclusive).

Moreover, for each $\phi_{ij}$, there is a set of conjunctive formulae, $\psi_{a_1}, \ldots, \psi_{a_q}$ such that $\phi_{ij} \leftrightarrow \psi_{a_1} \lor \ldots \lor \psi_{a_q}$. We can also refer to this set of ‘$\psi$-formulae’ as: $\{\psi_a \mid a \models \phi_{ij}\}$.

Given an assignment $a$ and a variable $x_i$, define $j(i, a)$ to be the unique $j$ such that $a \models \phi_{ij}$.
In order to prove that \( LHS \leftrightarrow RHS \), we will show that, for any \( a, \psi_a \land LHS \leftrightarrow \psi_a \land RHS \).

In fact, it is sufficient to show that, for any \( a, \psi_a \land LHS_i \leftrightarrow \psi_a \land RHS_i \), for some \( i \). There are two cases for the forms of \( LHS_i \) and \( RHS_i \) depending upon the value of \( i \).

Case 1 \( i \leq l \)

\[
LHS_i \overset{\text{def}}{=} \bigwedge_{k=1}^{m_i} (\alpha_{ik} \rightarrow \bigwedge_{v \in B_{ik}} x_{i,v}^f \land \bigwedge_{v \notin B_{ik}} \neg x_{i,v}^f) \land \\
\bigwedge_{j=1}^{r_i} (\neg \alpha_i \land \phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} x_{i,v}^f \land \bigwedge_{v \notin A_{ij}} \neg x_{i,v}^f)
\]

\[
RHS_i \overset{\text{def}}{=} \bigvee_{y=1}^{2^n} \left( \psi_{a_y} \land \left( \bigvee_{k=1}^{m_i} (\alpha_{ik} \land \bigwedge_{v \in B_{ik}} x_{i,v}^f \land \bigwedge_{v \notin B_{ik}} \neg x_{i,v}^f) \lor \\
(\neg \alpha_i \land \bigwedge_{v \in A_{ij}(i,a_y)} x_{i,v}^f \land \bigwedge_{v \notin A_{ij}(i,a_y)} \neg x_{i,v}^f) \right) \right)
\]

Take an assignment, \( a_y \).

\[
\psi_{a_y} \land LHS_i = \psi_{a_y} \land \left( \bigwedge_{k=1}^{m_i} (\alpha_{ik} \rightarrow \bigwedge_{v \in B_{ik}} x_{i,v}^f \land \bigwedge_{v \notin B_{ik}} \neg x_{i,v}^f) \land \\
\bigwedge_{j=1}^{r_i} (\neg \alpha_i \land \phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} x_{i,v}^f \land \bigwedge_{v \notin A_{ij}} \neg x_{i,v}^f) \right)
\]

Again, there are two cases. Either \( a_y \) makes \( \alpha_{ik} \) true for one and only one \( k \), in which case:

\[
\psi_{a_y} \land LHS_i = \psi_{a_y} \land \alpha_{ik} \land \bigwedge_{v \in B_{ik}} x_{i,v}^f \land \bigwedge_{v \notin B_{ik}} \neg x_{i,v}^f
\]

And in this case, we also get:

\[
\psi_{a_y} \land RHS_i = \psi_{a_y} \land \alpha_{ik} \land \bigwedge_{v \in B_{ik}} x_{i,v}^f \land \bigwedge_{v \notin B_{ik}} \neg x_{i,v}^f
\]

Or \( a_y \rightarrow \neg \alpha_i \), in which case \( a_y \) makes each \( \phi_{ij} \) false except one, namely \( \phi_{ij(i,a)} \):

\[
\psi_{a_y} \land LHS_i = \psi_{a_y} \land \phi_{ij(i,a)} \land \bigwedge_{v \in A_{ij}(i,a_y)} x_{i,v}^f \land \bigwedge_{v \notin A_{ij}(i,a_y)} \neg x_{i,v}^f
\]

And, in this second case, we also get:

\[
\psi_{a_y} \land RHS_i = \psi_{a_y} \land \phi_{ij(i,a)} \land \bigwedge_{v \in A_{ij}(i,a_y)} x_{i,v}^f \land \bigwedge_{v \notin A_{ij}(i,a_y)} \neg x_{i,v}^f
\]
**Propositional Logic Denotation of SMV**

Case \((l + 1) \leq i \leq n\)

\[
\text{LHS}_i \overset{\text{def}}{=} \bigwedge_{j=1}^{r_i} \left( \phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} \psi_{i,v} \wedge \bigwedge_{v \not\in A_{ij}} \neg x_{i,v}^j \right)
\]

\[
\text{RHS}_i \overset{\text{def}}{=} \bigvee_{y=1}^{2^n} \left( \psi_{a_y} \wedge \bigwedge_{v \in A_{ij(i\wedge y)}} x_{i,v}^j \wedge \bigwedge_{v \not\in A_{ij(i\wedge y)}} \neg x_{i,v}^j \right)
\]

Here, \(\text{LHS}_i\) and \(\text{RHS}_i\) are the same as in the proof of Theorem 4.2 and we prove that they are equivalent similarly.

### 4.6 What do the Update Axioms mean for SMV Feature Integration?

It is important to realise that we are not saying that feature integration always satisfies the eight rationality postulates for update, or even that this should be the case. Rather, we are saying that SMV feature integration in particular always satisfies (U1)-(U8) for our propositional logic denotation (although we do suspect that this might also be the case for some of the other feature constructs in the literature). It is interesting and important then, to examine what the axioms mean in this context. See Section 3.2 for a list of the postulates and see the proof of Theorem 4.1 for the way in which the eight axioms are translated into the context of SMV feature integration.

**(U1)** This axiom specifies that the feature is always successfully integrated into the updated system and this is indeed the case with feature integration in SMV. It is arguable that this is not always desirable. For example, if there is a set of base system properties which a feature is not permitted to disrupt, then feature integration could fail when a feature would violate one of these core properties.

**(U2)** From the point of view of SMV, this postulate states that if a system behaves as though a feature has already been integrated into it, then upon integration of the feature, the system remains unchanged. This property seems desirable of feature integration in general.
Propositional Logic Denotation of SMV

(U3) Satisfiability of $[P]$, $[F]$ and $[P + F]$ is addressed by this axiom. The way in which SMV case statements are evaluated means that a variable cannot be assigned contradictory values in a particular case. This is reflected in our abstract representation of SMV by means of the requirement that the cases themselves are covering and exclusive. Therefore, the denotations of either an SMV program or feature should be satisfiable and, of course, in this event, it makes sense that the updated program should be satisfiable. Again, it is difficult to think of a general feature integration example in which one would not wish this property to hold.

(U4) The principle of the irrelevance of syntax is as applicable to denotations of programs and features as it is to any other types of formulae. It simply stipulates that for two equivalent programs and features, the results of their respective updates should also be equivalent.

(U5) This axiom is concerned with minimal change to the base system in the context of composite update. It specifies that, when updating by two features, say $A$ and $B$, one should update by $A$ and then, as long as $B$ does not conflict with the result, simply expand by $B$, thereby retaining as much behaviour of the base system as possible.

The concept of making minimal changes to the base system is familiar in the context of system update. Of course, in reality there are other constraints which must also be taken into account such as cost and efficiency.

(U6) Again, this property is generally desirable of feature integration. It is an integrity constraint which states that, for two features, $A$ and $B$, if a system updated by $A$ implies $B$ and the same system updated by $B$ implies $A$, then both features yield an equivalent updated system.

(U7) This postulate applies only to systems which are represented by a complete formula. In fact, for our denotation of SMV, it is impossible for an entire system description to be specified by a complete formula, but from a theoretical point of view, it still merits consideration. In terms of models, and orderings upon them,
(U7) is stating that if a model $m$ of $\mu_1$ is minimal with respect to $\psi$ and it is also a model of $\mu_2$ which is minimal with respect to $\psi$, then $m$ is also a minimal model of $\mu_1 \lor \mu_2$ with respect to $\psi$. In other words, this axiom makes explicit the fact that models of $\mu_1$ and $\mu_2$ are ordered only once if there is only a single model of $\psi$.

(U8) This axiom is concerned with orderings on models once more. It ensures that each model of the base system is given independent consideration with regard to models of a feature.

The proof of Theorem 4.1 which we saw earlier in this chapter gives strong evidence of a correlation between SMV feature integration and update. However, it is unsatisfactory that axioms (U5), (U7) and (U8) could not be formulated in certain cases. Furthermore, Katsuno and Mendelzon defined update on two arbitrary propositional logic formulae, $\psi$ and $\mu$, whereas here we have defined $\psi$ and $\mu$ to be $[P]$ and $[F]$ respectively. In effect, then, we have reasoned about a special case and the definition involved is restrictive. Therefore, in Chapter 5 we generalise this result proving conclusively that feature integration is an operation of update. We can then make use of Katsuno and Mendelzon’s representation theorem.
Chapter 5

A $\diamond$ Operator for SMV Feature Integration

In this chapter, we define a $\diamond$ operator on arbitrary formulae $\psi$ and $\mu$ such that, when $\psi$ represents an SMV program and $\mu$ a feature, $\psi \diamond \mu$ specialises to SMV feature integration. We prove this behaviour formally and show that this $\diamond$ operator satisfies (U1)-(U8). Theorem 4.1 provided strong evidence of a relationship between feature integration and the update operation but the theorem only dealt with formulae which denoted SMV programs and features, it did not generalise to arbitrary formulae. Therefore, Chapter 4 essentially dealt with a special case. Chapter 5 strengthens the result considerably to show that there is a relationship between update and feature integration in the general case. Therefore, the work in this chapter gives irrefutable proof of the correlation. In addition, it enables us to draw upon Katsuno and Mendelson’s representation theorem, which will allow us to consider programs and features from a model-theoretic point of view in the future.

To use an analogy, in Chapter 4, it is as though we found a key (Theorem 4.1) which allowed us through the front door of a house and enabled us to access the downstairs rooms of the house, but not the upstairs (Katsuno and Mendelzon’s representation theorem), although we were able to see a flight of stairs which gave us evidence that the upstairs might exist. In Chapter 5, we have found a skeleton key (Theorems 5.1 and 5.2) which enables us to access the whole house, including the upstairs (the representation theorem). We need the representation theorem to hold so that we can begin to consider an ordering
on programs in relation to the base system.

In technical terms, then, we need to define a $\diamond$ operator in this chapter which works on any two formulae, but which specialises to feature integration when $\psi$ happens to represent a program and $\mu$ happens to represent a feature. Then we can draw upon the semantic machinery (the representation theorem) which comes with the $\diamond$ operator. This is what we do in this chapter.

We begin with some preliminaries in Section 5.1. This is followed by our definition of the $\diamond$ operator in Section 5.2 and a detailed explanation of exactly how it relates to feature integration. Then in the following two Sections, we give formal proofs that the $\diamond$ operator behaves as we claim it does and that it satisfies (U1)-(U8).

The results in this chapter were first published in [HR03].

### 5.1 Preliminaries

We assume a finite set $\text{Vars} = X \uplus X'$ of atomic propositions where, in terms of SMV programs, $X$ is a set of propositions representing the current values of variables and $X'$ is a set of primed propositions representing the next values of variables (this was explained in more detail in Chapter 4). Formulae are given by the usual propositional logic grammar:

$$\phi ::= x | \neg \phi | \phi_1 \land \phi_2 | \phi_1 \lor \phi_2 | \phi_1 \rightarrow \phi_2 | \phi_1 \leftrightarrow \phi_2 | \top | \bot$$

where $x \in \text{Vars}$.

**Definition 5.1** A formula $\phi$ is complete over a set of propositions $U \subseteq \text{Vars}$ iff for all $u \in U$

$$\phi \models u \text{ or } \phi \models \neg u$$

If the set $U$ is unspecified, we assume $U = \text{Vars}$.

Furthermore, $\phi$ is complete iff it has a unique model which we denote by $m_\phi$.

**Definition 5.2** We denote the set of models associated with a formula $\phi$ by $\text{Mod}(\phi)$. Hence, for complete $\phi$, $\text{Mod}(\phi) = \{m_\phi\}$. 
Observation 5.3 Any formula $\phi$ may be rewritten as an equivalent formula which consists of a disjunction of complete formulae $\phi_1, \ldots, \phi_n$. There is a one-to-one correspondence between these disjuncts and models of $\phi$. In fact, $\text{Mod}(\phi) = \{m_{\phi_1}, \ldots, m_{\phi_n}\}$.

Definition 5.4 If $\phi$ is a formula, $v \in \text{Vars}$ and $U \subseteq \text{Vars}$ then:

1. $\exists v. \phi \overset{\text{def}}{=} \phi[\top/v] \lor \phi[\bot/v]$

2. $\exists U. \phi \overset{\text{def}}{=} \exists u_1 \ldots \exists u_n. \phi$
where $U = \{u_1, \ldots, u_n\}$

3. $\phi|_U \overset{\text{def}}{=} \exists (\text{Vars} \setminus U). \phi$

$\phi|_U$ is called the restriction of $\phi$ to $U$.

Observation 5.5 For a complete formula $\phi$,

$$\phi \leftrightarrow \phi\big|_X \land \phi\big|_{X'}$$

In Definition 5.4 above, we defined the restriction of a formula to specific variables. This is crucial to the $\circ$ operator definition because we need the facility to reason about current and next variables separately. The idea is that if we restrict a formula $\phi$ to propositions in a set $U$, then we assign values to propositions not in $U$ in order to make $\phi$ true. This results in a formula $\phi|_U$ which is in terms of propositions in $U$ only, and which is therefore the restriction of $\phi$ to $U$.

Example 5.1

$$\phi = p \land q \land \neg r$$

$$U = \{q\}$$

$$\phi|_U = (\top \land q \land \neg \top) \lor (\top \land q \land \bot) \lor$$

$$\bot \land q \land \bot) \lor (\bot \land q \land \bot)$$

$$= \bot \lor q \lor \bot \lor \bot$$

$$= q$$
From Example 5.1, we can see that if we restrict to a particular proposition in a formula which consists of a large conjunction, we succeed in extracting the value of the proposition in question. This is a property which we exploit in Definition 5.9.

**Definition 5.6** If $m$ is a model and $x$ a proposition, $m^x$ is the model $m$ with the opposite value of $x$.

Now let $M$ be a set of models.

**Definition 5.7** A model ordering is a family of relations $\{\leq_m \mid m \in M\}$ such that:

- $\leq_m$ is reflexive and transitive.
- for all $m, n \in M$, $m <_m n$ iff $n \neq m$.
- $n \leq_m n^x$, for all $m, n$ agreeing on proposition $x$.

**Example 5.2** The following are examples of model orderings which satisfy the constraints as given above in Definition 5.7.

1. Winslett (see Example 3.2).
2. Dalal (see Example 3.2).

**Definition 5.8** $\blacklozenge$ is a black diamond operator if there is a model ordering $\leq$ such that, for two formulae $\psi$ and $\mu$:

$$\text{Mod}(\psi \blacklozenge \mu) = \bigcup_{m \in \text{Mod}(\psi)} \text{Min}_{\leq_m}(\text{Mod}(\mu))$$

The black diamond operator as defined in terms of Winslett and Dalal’s model orderings, is also an update operator as defined by Katsuno and Mendelzon in [KM92]. For each model of $\psi$, it selects the closest models of $\mu$. 
5.2 Definition of the \( \diamond \) Operator

The update operator \( \diamond \) is defined on two arbitrary propositional logic formulae, \( \psi \) and \( \mu \), such that if \( \psi = [P] \) and \( \mu = [F] \) (for some SMV program \( P \) and feature \( F \)), then \( \psi \diamond \mu = [P + F] \).

The reason why both Winslett or Dalal’s orderings are suitable in Definition 5.8 is that they both yield the same results when \( \psi = [P] \) and \( \mu = [F] \). This is because Winslett and Dalal’s orderings satisfy all the required properties as given in Definition 5.7, and the proofs in Sections 5.3 and 5.4 are dependent only upon these general constraints, as opposed to technical details of any particular ordering.

For all other formulae \( \psi \) and \( \mu \), we simply require \( \diamond \) to satisfy (U1)-(U8) and, from Katsuno and Mendelzon’s representation theorem as presented in Chapter 3, we know this to be the case for both Winslett and Dalal’s orderings.

**Definition 5.9** \( \diamond \) is a white diamond operator if there is a black diamond operator \( \bullet \) such that:

- If \( \psi \) is complete,
  \[
  \psi \diamond \mu \overset{\text{def}}{=} \begin{cases} 
  \psi \bullet (\psi|_X \land \mu) & \text{if } \psi|_X \land \mu \not\rightarrow \bot \\
  \psi \bullet \mu & \text{otherwise}
  \end{cases}
  \]

- If \( \psi \) is not complete, we use Observation 5.3 to transform \( \psi \) into an equivalent disjunctive formula, \( \psi_1 \lor \ldots \lor \psi_n \) where each \( \psi_i \) is complete. Then we define:
  \[
  \psi \diamond \mu \overset{\text{def}}{=} (\psi_1 \diamond \mu) \lor \ldots \lor (\psi_n \diamond \mu)
  \]

Note that when \( \psi \) and \( \mu \) are a program and a feature respectively, and \( \psi \leftrightarrow \psi_1 \lor \ldots \lor \psi_n \), \( \psi|_X \land \mu \not\rightarrow \bot \) will always be the case. This is because, when \( \mu \) is the denotation of a feature, it will consist of a large conjunction of implications in which \( X \)-values appear only in the antecedents. Therefore, \( \mu \) will not contradict the \( X \)-values specified by \( \psi_i \).

So why do we need the second case in our definition? Well, we are specifying the \( \diamond \) operator on arbitrary formulae \( \psi \) and \( \mu \) and, for some formulae, specifically those in which \( \mu \) contradicts the \( X \)-values specified by \( \psi_i \) for some \( i \), (U1)-(U8) will be violated.
For example, take (U3): If \( \psi \) and \( \mu \) are satisfiable, then \( \psi \odot \mu \) is satisfiable. Consider
\[
\psi = ( (x = 1) \land x'_1 \land x'_2 ) \quad \text{and} \quad \mu = ( x = 2 ).
\]
Both \( \psi \) and \( \mu \) are satisfiable but \( \psi \odot (\psi |_x \land \mu) = \bot \) so (U3) is violated. Hence, we need the second part of our definition to cover such a case.

It is important to note why we update \( \psi \) by \( \psi|_x \land \mu \) when \( \psi = [P] \) and \( \mu = [F] \). The reason is associated with the partition of variables into the two sets, \( X \) and \( X' \). \( \psi \) is complete so it represents a single current state and all of its possible successor states. We have to extract this current state by taking \( \psi|_x \) and then this is conjoined with \( \mu \) in order to ascertain what the feature specifies the successor states should be for this current state, that is, if the feature does indeed apply at all in this case.

5.3 Proof of Correct Behaviour

Theorem 5.1 (below) concerns our claim that the \( \odot \) operator of Definition 5.9 performs the operation of SMV feature integration when \( \psi \) and \( \mu \) are a program and a feature respectively. It is preceded by necessary definitions and Lemma 5.1, which is also necessary for the proof of the theorem.

In the following lemma, theorem and respective proofs we use the equivalent denotation of an SMV program \( P \), as given in Theorem 4.2 in Chapter 4:

\[
[P] \overset{\text{def}}{=} \bigvee_{y=1}^{2^n} \big( \psi_{\alpha_y} \land \bigwedge_{i=1}^{n} \left( \bigwedge_{v \in A_{ij(i,s_y)}} x'_{i,v} \land \bigwedge_{v \not\in A_{ij(i,s_y)}} \neg x'_{i,v} \right) \big)
\]

where \( x'_{i,v} \in X'_P \).

We use the denotation of an SFI feature as given in Definition 4.7 in Chapter 4:

\[
[F] \overset{\text{def}}{=} \bigwedge_{i=1}^{l} \bigwedge_{k=1}^{m_i} ( \alpha_{ik} \rightarrow \bigwedge_{v \in B_{ik}} x'_{i,v} \land \bigwedge_{v \not\in B_{ik}} \neg x'_{i,v} )
\]

where \( x'_{i,v} \in X'_F \).

We use the equivalent denotation of a featured program, \( P + F \), as given in Theorem 4.3
in Chapter 4:

\[ P + F \] = \bigvee_{y=1}^{2^n} \left( \psi_{ay} \land \left( \bigwedge_{i=1}^{l} \left( \bigwedge_{k=1}^{m_i} \left( \alpha_{ik} \land x'_{i,v} \land \neg x'_{i,v} \right) \lor \left( \neg \alpha_i \land \bigwedge_{v \in A_{ij,(i,a,y)}} x'_{i,v} \land \bigwedge_{v \not\in A_{ij,(i,a,y)}} \neg x'_{i,v} \right) \right) \bigwedge_{i=l+1}^{n} \left( \bigwedge_{v \in A_{ij,(i,a,y)}} x'_{i,v} \land \bigwedge_{v \not\in A_{ij,(i,a,y)}} \neg x'_{i,v} \right) \right) \land \right)

where \( \alpha_i = \bigvee_{k=1}^{m_i} \alpha_{ik} \) and \( x'_{i,v} \in X'_{P+F} \).

**Lemma 5.1 (Denotation of a Program Updated by a Feature, \([P] \circ \[F]\))**

Let \([P]\) and \([F]\) be defined as above, and \(\circ\) as in Definition 5.9. Then:

\[ [P] \circ [F] = \bigvee_{y=1}^{2^n} \left( \psi_{ay} \land \left( \bigwedge_{i=1}^{l} \left( \bigwedge_{k=1}^{m_i} \left( \alpha_{ik} \land x'_{i,v} \land \neg x'_{i,v} \right) \lor \left( \neg \alpha_i \land \bigwedge_{v \in A_{ij,(i,a,y)}} x'_{i,v} \land \bigwedge_{v \not\in A_{ij,(i,a,y)}} \neg x'_{i,v} \right) \right) \bigwedge_{i=l+1}^{n} \left( \bigwedge_{v \in A_{ij,(i,a,y)}} x'_{i,v} \land \bigwedge_{v \not\in A_{ij,(i,a,y)}} \neg x'_{i,v} \right) \right) \lor \right)

**Proof of Lemma 5.1.** This lemma shows the denotation for \([P]\) updated by \([F]\), that is, \([P] \circ [F]\), in which, for each valuation \(\psi_{ay}\) of propositions in \(X\), there is a disjunct of the form \([P]_y \lor ([P]_y \land [F])\). Recall that each \([P]_y\) is complete and therefore corresponds to a model of \([P]\). We explained before why \([P]_y \land [F] \not\Rightarrow \bot\) for all \(y\), and hence why, by definition of \(\circ\), \([P]_y \circ [F] = [P]_y \lor ([P]_y \land [F])\). 

**Theorem 5.1** Consider the propositional logic denotations for an SMV program \(P\) and a feature \(F\) shown above, and the \(\circ\) operator of Definition 5.9.

\[ [P + F] \iff [P] \circ [F] \]

**Proof of Theorem 5.1.** In this proof, we refer to \([P]\), \([F]\), \([P + F]\) and \([P] \circ [F]\) as shown above.

Let \(\theta_{+y}\) be the \(y\)th disjunct of \([P + F]\) and \(\theta_{-y}\) the \(y\)th disjunct of \([P] \circ [F]\), i.e.

\[ [P + F] = \bigvee_{y=1}^{2^n} \theta_{+y} \text{ and } [P] \circ [F] = \bigvee_{y=1}^{2^n} \theta_{-y} \]

\(\theta_{+y}\) and \(\theta_{-y}\) are complete over \(\text{Vars}\).
For all \( \psi_{a_y} \), it is sufficient to show that \( \theta_+ \models \theta_{o_y} \).

Further, from Observation 5.5, we can see that it is sufficient to show that \( \theta_+ | x \models \theta_{o_y} | x \)
and that \( \theta_+ | x' \models \theta_{o_y} | x' \).

Take any \( \psi_{a_y} \), we can see that \( \theta_+ | x \models \theta_{o_y} | x \models \psi_{a_y} \).

Now we need to show that \( \theta_+ | x' \models \theta_{o_y} | x' \).

With each \( x_i \in X \) is associated a set of propositions \( N_{x_i} \subseteq X' \) such that \( N_{x_i} = \{ x_i' | v \in \text{type}(x_i) \} \). For each \( x_i \), we need to show that \( \theta_+ | N_{x_i} \models \theta_{o_y} | N_{x_i} \).

Take any \( x_i \in \text{Vars} \). There are two cases:

1. Case \( x_i \in \{ x_1, \ldots, x_l \} \) There are two further cases:

   a. Case \( \psi_{a_y} \rightarrow \alpha_{i_k} \), for some \( k \)

   In this case, it can be shown that

   \[
   \theta_+ | N_{x_i} \models \theta_{o_y} | N_{x_i} \iff \bigwedge_{v \in B_{i_k}} x_i' v \land \bigwedge_{v \not\in B_{i_k}} \neg x_i' v
   \]

   In the case of \( \Box P \land \Box F \), it is clear from the first line of its definition above, that

   \[
   \theta_+ | N_{x_i} \iff \bigwedge_{v \in B_{i_k}} x_i' v \land \bigwedge_{v \not\in B_{i_k}} \neg x_i' v
   \]

   To get \( \theta_{o_y} | N_{x_i} \iff \bigwedge_{v \not\in B_{i_k}} x_i' v \land \bigwedge_{v \not\in B_{i_k}} \neg x_i' v \), we observe that

   \[
   \psi_{a_y} \land \bigwedge_{i=1}^{l} \bigwedge_{k=1}^{m_i} (\alpha_{i_k} \rightarrow \bigwedge_{v \in B_{i_k}} x_i' v \land \bigwedge_{v \not\in B_{i_k}} \neg x_i' v)
   \]

   is complete over \( N_{x_i} \) and use the following lemma:

   If \( \mu \) is complete over \( U \subseteq \text{Vars} \) then \( (\psi \bullet \mu) | U = \mu | U \)

   This follows from (U1). \( \psi \bullet \mu \Rightarrow \mu \) implies \( (\psi \bullet \mu) | U \Rightarrow \mu | U \), and, because \( \mu \) is complete over \( U \), the latter implication can only hold if \( (\psi \bullet \mu) | U = \mu | U \).

   b. Case \( \psi_{a_y} \rightarrow \neg \alpha_{i_k} \)

   In this case, it can be shown that

   \[
   \theta_+ | N_{x_i} \models \theta_{o_y} | N_{x_i} \iff \bigwedge_{v \in A_{ij}(i,a_y)} x_i' v \land \bigwedge_{v \not\in A_{ij}(i,a_y)} \neg x_i' v
   \]
In the case of \( \llbracket P + F \rrbracket \), it is clear from the second line of its definition above, that \( \theta_{+y} |_{N_{z_i}} \leftrightarrow \bigwedge_{v \in A_{ij(i,v,y)}} x_{i,v} \lor \bigwedge_{v \notin A_{ij(i,v,y)}} \neg x_{i,v} \).

To get \( \theta_{\neg y} |_{N_{z_i}} \leftrightarrow \bigwedge_{v \in A_{ij(i,v,y)}} x_{i,v} \lor \bigwedge_{v \notin A_{ij(i,v,y)}} \neg x_{i,v} \), we observe that

\[
\psi_{\neg y} \land \bigwedge_{i=1}^{n} \left( \bigwedge_{v \in A_{ij(i,v,y)}} x_{i,v} \lor \bigwedge_{v \notin A_{ij(i,v,y)}} \neg x_{i,v} \right)
\]

is complete and that \( \mu |_{N_{z_i}} = \top \). Then we use Lemma 5.2, a proof of which appears at the end of this proof.

**Lemma 5.2**

If \( \psi \) is complete, \( U \subseteq Vars \), and \( \mu |_{U} = \top \), then \( \left( \psi \circ \mu \right) |_{U} = \psi |_{U} \)

2. Case \( x_i \in \{ x_{i+1}, \ldots, x_n \} \). In this case, it can be shown that

\[
\theta_{+y} |_{N_{z_i}} \leftrightarrow \theta_{\neg y} |_{N_{z_i}} \leftrightarrow \bigwedge_{v \in A_{ij(i,v,y)}} x_{i,v} \land \bigwedge_{v \notin A_{ij(i,v,y)}} \neg x_{i,v}
\]

In the case of \( \llbracket P + F \rrbracket \), it is clear from the third line of its definition above, that

\( \theta_{+y} |_{N_{z_i}} \leftrightarrow \bigwedge_{v \in A_{ij(i,v,y)}} x_{i,v} \lor \bigwedge_{v \notin A_{ij(i,v,y)}} \neg x_{i,v} \).

In the case of \( \llbracket P \rrbracket \circ \llbracket F \rrbracket \), we can apply Lemma 5.2 again to show that \( \theta_{\neg y} |_{N_{z_i}} \leftrightarrow \bigwedge_{v \in A_{ij(i,v,y)}} x_{i,v} \lor \bigwedge_{v \notin A_{ij(i,v,y)}} \neg x_{i,v} \).

**Proof of Lemma 5.2.**

**Notation.** If \( m \) is a model and \( x \) a proposition, \( m^x \) is the model \( m \) with the opposite value of \( x \).

**Requirement on \( \leq \).** Suppose \( m, n \) agree on \( x \). Then \( n \leq m \ n^x \).

**Lemma 5.3** Suppose that \( \psi \) is complete, i.e. \( \text{Mod}(\psi) = m_\psi \). Then \( \text{Mod}(\exists x. \psi) = \{ m_\psi, m_\psi^x \} \).

**Lemma 5.4** Suppose that \( \text{Mod}(\psi) = \{ m_1, \ldots, m_t \} \). Then \( \text{Mod}(\exists x. \psi) = \{ m_1, \ldots, m_t \} \cup \{ m_1^x, \ldots, m_t^x \} \).
Lemma 5.5 If \( \psi \) is complete then \( \exists x.(\psi \circ \mu) = \exists x.\psi \circ \exists x.\mu. \)

A proof of Lemma 5.5 appears at the end of this proof.

Corollary. If \( \psi \) is complete, then \( (\psi \circ \mu)|_U = \psi|_U \circ \mu|_U. \)

Corollary. If \( \psi \) is complete, \( U \subseteq \text{Vars} \), and \( \mu|_U = \top \), then \( (\psi \circ \mu)|_U = \psi|_U. \)

Proof of Lemma 5.5.

Let \( \text{Mod}(\mu) = \{m_1, \ldots, m_k\}. \)

\( \text{Mod}(\exists x.\psi \circ \exists x.\mu) = \text{Min}_{\leq m_\psi} (\text{Mod}(\exists x.\mu)) \cup \text{Min}_{\leq m_\psi} (\text{Mod}(\exists x.\mu)) \)

\[ = \text{Min}_{\leq m_\psi} (\{m_1, \ldots, m_k, m_1^x, \ldots, m_k^x\}) \cup \text{Min}_{\leq m_\psi} (\{m_1, \ldots, m_k, m_1^x, \ldots, m_k^x\}) \]

\[ = \{m, m^x | m \in \text{Min}_{\leq m_\psi} (\{m_1, \ldots, m_k\}) \} \]

\[ = \text{Mod}(\exists x.(\psi \circ \mu)). \]

5.4 Proof of Update Axioms

In this section, we initially prove Lemma 5.6 which shows that (U1)-(U8) hold in the case where the knowledge base \( \psi \) is complete. We then go on to prove our principal theorem which specifies that (U1)-(U8) hold in the general case, where \( \psi \) may be incomplete.

Lemma 5.6 In the case where \( \psi \) is complete, the new \( \circ \) operator as defined in Definition 5.9 satisfies (U1)-(U8).

Proof of Lemma 5.6.

(U1) \( \psi \circ \mu \) implies \( \mu. \)

Assume \( \psi \circ \mu \)

Case \( \psi|_X \wedge \mu \not\rightarrow \bot \)

Required to prove that \( \psi \circ (\psi|_X \wedge \mu) \Rightarrow \mu \)

We know that \( \circ \) satisfies (U1), so we have \( \psi|_X \wedge \mu \) and thus we get \( \mu. \)
Case \( \psi|_X \land \mu \rightarrow \bot \)

Required to prove that \( \psi \land \mu \Rightarrow \mu \)

We know that \( \diamond \) satisfies (U1), so we have \( \mu \).

(U2) If \( \psi \) implies \( \mu \) then \( \psi \land \mu \) is equivalent to \( \psi \).

Assume \( \psi \rightarrow \mu \)

Case \( \psi|_X \land \mu \not\rightarrow \bot \)

Required to prove that \( \psi \land (\psi|_X \land \mu) \Leftrightarrow \psi \)

\( \psi \rightarrow \mu \) and \( \psi \rightarrow \psi|_X \) so \( \psi \rightarrow \psi|_X \land \mu \)

\( \diamond \) satisfies (U2) so we get \( \psi \land (\psi|_X \land \mu) \Leftrightarrow \psi \)

Case \( \psi|_X \land \mu \rightarrow \bot \)

Required to prove that \( \psi \land \mu \Leftrightarrow \psi \)

\( \psi \rightarrow \mu \) and \( \diamond \) satisfies (U2) so we get \( \psi \land \mu \Leftrightarrow \psi \).

(U3) If both \( \psi \) and \( \mu \) are satisfiable then \( \psi \land \mu \) is also satisfiable.

Assume \( \psi \) and \( \mu \) are satisfiable.

Case \( \psi|_X \land \mu \not\rightarrow \bot \)

Required to prove that \( \psi \land (\psi|_X \land \mu) \) is satisfiable.

\( \psi \) is satisfiable and this case means that \( \psi|_X \land \mu \) is satisfiable.

Since \( \diamond \) satisfies (U3), we can see that \( \psi \land (\psi|_X \land \mu) \) is satisfiable.

Case \( \psi|_X \land \mu \rightarrow \bot \)

Required to prove that \( \psi \land \mu \) is satisfiable.

Again, the fact that \( \diamond \) satisfies (U3) proves this.

(U4) If \( \psi_1 \Leftrightarrow \psi_2 \) and \( \mu_1 \Leftrightarrow \mu_2 \) then \( \psi_1 \land \mu_1 \Leftrightarrow \psi_2 \land \mu_2 \).

Assume \( \psi_1 \Leftrightarrow \psi_2 \) and \( \mu_1 \Leftrightarrow \mu_2 \).

\( \psi_1[x/v] \Leftrightarrow \psi_2[x/v] \)

and therefore \( \psi_1|_X \Leftrightarrow \psi_2|_X \)

and therefore \( \psi_1|_X \land \mu_1 \Leftrightarrow \psi_2|_X \land \mu_2 \)
Hence, $\psi_1|_X \land \mu_1 \rightarrow \bot$ iff $\psi_2|_X \land \mu_2 \rightarrow \bot$

Moreover, $\psi_1 \bullet \mu_1 \leftrightarrow \psi_2 \bullet \mu_2$ because $\bullet$ satisfies (U4)

Case $\psi_1|_X \land \mu_1 \not\rightarrow \bot$ and $\psi_2|_X \land \mu_2 \not\rightarrow \bot$

Required to prove that $\psi_1 \bullet (\psi_1|_X \land \mu_1) \leftrightarrow \psi_2 \bullet (\psi_2|_X \land \mu_2)$

We know that $\psi_1 \leftrightarrow \psi_2$ and $\psi_1|_X \land \mu_1 \leftrightarrow \psi_2|_X \land \mu_2$.

Therefore, we use the fact that $\bullet$ satisfies (U4) to obtain the desired conclusion.

Case $\psi_1|_X \land \mu_1 \rightarrow \bot$ and $\psi_2|_X \land \mu_2 \rightarrow \bot$

Required to prove that $\psi_1 \bullet \mu_1 \leftrightarrow \psi_2 \bullet \mu_2$. Again, we use the fact that $\bullet$ satisfies (U4).

These are the only two cases which are possible, given the assumptions.

(U5) $\psi \circ \mu \land \phi$ implies $\psi \circ (\mu \land \phi)$.

Assume $(\psi \circ \mu) \land \phi$

Case $\psi|_X \land \mu \not\rightarrow \bot$

Case $\psi|_X \land \mu \land \phi \not\rightarrow \bot$

Required to prove that $(\psi \bullet (\psi|_X \land \mu)) \land \phi \Rightarrow \psi \bullet (\psi|_X \land \mu \land \phi)$

To prove this, we simply use the fact that $\bullet$ satisfies (U5).

Case $\psi|_X \land \mu \land \phi \rightarrow \bot$

Required to prove that $(\psi \bullet (\psi|_X \land \mu)) \land \phi \Rightarrow \psi \bullet (\psi|_X \land \mu \land \phi)$

$\bullet$ satisfies (U5) so $\psi \bullet (\psi|_X \land \mu) \land \phi \Rightarrow \psi \bullet (\psi|_X \land \mu \land \phi)$.

But in this case we know that $\psi|_X \land \mu \land \phi \rightarrow \bot$.

$\bullet$ satisfies (U1) so $\psi \bullet (\psi|_X \land \mu \land \phi) \rightarrow \bot$.

Therefore, $\psi \bullet (\psi|_X \land \mu) \land \phi \rightarrow \bot$ and therefore implies $\psi \bullet (\mu \land \phi)$.

Case $\psi|_X \land \mu \rightarrow \bot$

In this case, $\psi|_X \land \mu \land \phi \rightarrow \bot$.

Required to prove that $(\psi \bullet \mu) \land \phi \Rightarrow \psi \bullet (\mu \land \phi)$.

Again, we use the fact that $\bullet$ satisfies (U5).
(U6) If \( \psi \odot \mu_1 \) implies \( \mu_2 \) and \( \psi \odot \mu_2 \) implies \( \mu_1 \) then \( \psi \odot \mu_1 \Leftrightarrow \psi \odot \mu_2 \).

Assume \( \psi \odot \mu_1 \rightarrow \mu_2 \) and \( \psi \odot \mu_2 \rightarrow \mu_1 \)

\begin{itemize}
  \item Case \( \psi|_X \land \mu_1 \not\rightarrow \bot \)
     \begin{itemize}
        \item Case \( \psi|_X \land \mu_2 \not\rightarrow \bot \)
            \begin{itemize}
                \item Required to prove that \( \psi \odot (\psi|_X \land \mu_1) \Leftrightarrow \psi \odot (\psi|_X \land \mu_2) \)
                \item We know that \( (\psi \odot (\psi|_X \land \mu_1)) \rightarrow \mu_2 \)
                \item We also know that \( (\psi \odot (\psi|_X \land \mu_1)) \rightarrow \psi|_X \) (because \( \odot \) satisfies (U1)).
                \item Therefore, \( (\psi \odot (\psi|_X \land \mu_1)) \rightarrow (\psi|_X \land \mu_2) \)
                \item We reason likewise to conclude that \( (\psi \odot (\psi|_X \land \mu_2)) \rightarrow (\psi|_X \land \mu_1) \)
                \item Now, because \( \odot \) satisfies (U6), we can conclude that \( \psi \odot (\psi|_X \land \mu_1) \Leftrightarrow \psi \odot (\psi|_X \land \mu_2) \)
            \end{itemize}
     \end{itemize}
  \item Case \( \psi|_X \land \mu_2 \rightarrow \bot \)
     \begin{itemize}
        \item Since the assumption that \( \psi \odot \mu_1 \rightarrow \mu_2 \) holds, this case is impossible, because from the assumption it follows that \( \psi \odot (\psi|_X \land \mu_1) \rightarrow \mu_2 \) and as \( \odot \) satisfies (U1), we know that \( \psi \odot (\psi|_X \land \mu_1) \rightarrow \psi|_X \). Therefore \( \psi \odot (\psi|_X \land \mu_1) \rightarrow (\psi|_X \land \mu_2) \).
        \item Hence \( \psi \odot (\psi|_X \land \mu_1) \rightarrow \bot \) (because of the case in consideration), but this implies that either \( \psi = \bot \) or that \( \psi|_X \land \mu_1 \rightarrow \bot \), both of which contradict the fact that \( \psi|_X \land \mu_1 \not\rightarrow \bot \).
     \end{itemize}
  \item Case \( \psi|_X \land \mu_1 \rightarrow \bot \)
     \begin{itemize}
        \item Case \( \psi|_X \land \mu_2 \not\rightarrow \bot \)
            \begin{itemize}
                \item This case is impossible for the inverse of the reasons which rendered the previous case impossible.
            \end{itemize}
        \item Case \( \psi|_X \land \mu_2 \rightarrow \bot \)
            \begin{itemize}
                \item Required to prove that \( \psi \odot \mu_1 \Leftrightarrow \psi \odot \mu_2 \)
                \item From the assumptions, we have \( \psi \odot \mu_1 \rightarrow \mu_2 \) and \( \psi \odot \mu_2 \rightarrow \mu_1 \)
                \item As \( \odot \) satisfies (U6), we may draw the desired conclusion.
            \end{itemize}
     \end{itemize}
\end{itemize}

(U7) If \( \psi \) is complete then \( (\psi \odot \mu_1) \land (\psi \odot \mu_2) \) implies \( \psi \odot (\mu_1 \lor \mu_2) \).
Assume $\psi$ is complete.

Case $\psi_X \land \mu_1 \not\rightarrow \bot$

Case $\psi_X \land \mu_2 \not\rightarrow \bot$

Required to prove that:

$$(\psi \lor (\psi_X \land \mu_1)) \land (\psi \lor (\psi_X \land \mu_2)) \Rightarrow \psi \lor (\psi_X \land (\mu_1 \lor \mu_2))$$

(5.1)

(5.1) may be rewritten as follows:

$$(\psi \lor (\psi_X \land \mu_1)) \land (\psi \lor (\psi_X \land \mu_2)) \Rightarrow \psi \lor ((\psi_X \land \mu_1) \lor (\psi_X \land \mu_2))$$

(5.2)

$\lor$ satisfies (U7), so the implication in (5.2) holds.

Case $\psi_X \land \mu_2 \rightarrow \bot$

Required to prove that:

$$(\psi \lor (\psi_X \land \mu_1)) \land (\psi \lor \mu_2) \Rightarrow \psi \lor (\psi_X \land (\mu_1 \lor \mu_2))$$

(5.3)

The antecedent of (5.3) results in a contradiction because it implies $\psi_X \land \mu_2$, therefore the implication is vacuously true.

Case $\psi_X \land \mu_1 \rightarrow \bot$

Case $\psi_X \land \mu_2 \not\rightarrow \bot$

We use the inverse reasoning of that used in the previous case.

Case $\psi_X \land \mu_2 \rightarrow \bot$

Required to prove that:

$$(\psi \lor \mu_1) \land (\psi \lor \mu_2) \Rightarrow \psi \lor (\mu_1 \lor \mu_2)$$

(5.4)

As $\lor$ satisfies (U7), we can see that the implication in (5.4) holds.

(U8) $(\psi_1 \lor \psi_2) \lor \mu \leftrightarrow (\psi_1 \lor \mu) \lor (\psi_2 \lor \mu)$.

In order for $\psi_1 \lor \psi_2$ to be complete, it must be the case that $\psi_1 \leftrightarrow \psi_2$, in which case $(\psi_1 \lor \mu) \lor (\psi_2 \lor \mu)$ is trivial.
**Theorem 5.2** The new $\diamond$ operator as defined in Definition 5.9 satisfies (U1)-(U8).

**Proof of Theorem 5.2.** Lemma 5.6 proves this is the case where $\psi$ is complete so here, we consider only the case where $\psi$ is not complete. In the proofs below, we make use of the equivalence given in Observation 5.3. That is, $\psi \leftrightarrow (\psi_1 \lor \ldots \lor \psi_n)$ where each $\psi_i$ is complete.

**(U1)** $\psi \diamond \mu$ implies $\mu$.

$$\psi \diamond \mu = (\psi_1 \lor \ldots \lor \psi_n) \diamond \mu \quad (5.5)$$

$$\overset{\text{def}}{=} (\psi_1 \diamond \mu) \lor \ldots \lor (\psi_n \diamond \mu) \quad (5.6)$$

From the proof of (U1) in Lemma 5.6, we know that, for each $\psi_i$, $\psi_i \diamond \mu \rightarrow \mu$.

Therefore, $\psi \diamond \mu \rightarrow \mu$.

**(U2)** If $\psi$ implies $\mu$ then $\psi \diamond \mu$ is equivalent to $\psi$.

$$\psi \rightarrow \mu = (\psi_1 \lor \ldots \lor \psi_n) \rightarrow \mu \quad (5.7)$$

$$= (\psi_1 \rightarrow \mu) \land \ldots \land (\psi_n \rightarrow \mu) \quad (5.8)$$

From the proof of (U2) in Lemma 5.6, we know that, for each $\psi_i$, $\psi_i \rightarrow \mu \Rightarrow \psi_i \diamond \mu \leftrightarrow \psi_i$. So (5.8) implies:

$$\left( (\psi_1 \diamond \mu \leftrightarrow \psi_1) \land \ldots \land (\psi_n \diamond \mu \leftrightarrow \psi_n) \right) \quad (5.9)$$

$$\Rightarrow (\psi_1 \diamond \mu) \lor \ldots \lor (\psi_n \diamond \mu) \leftrightarrow (\psi_1 \lor \ldots \lor \psi_n) \quad (5.10)$$

$$\overset{\text{def}}{=} (\psi_1 \lor \ldots \lor \psi_n) \diamond \mu \leftrightarrow (\psi_1 \lor \ldots \lor \psi_n) \quad (5.11)$$

$$= \psi \diamond \mu \leftrightarrow \psi \quad (5.12)$$

**(U3)** If both $\psi$ and $\mu$ are satisfiable then $\psi \diamond \mu$ is also satisfiable.

Assume that $\psi$ and $\mu$ are satisfiable.

$$\psi \diamond \mu = (\psi_1 \lor \ldots \lor \psi_n) \diamond \mu \quad (5.13)$$

$$\overset{\text{def}}{=} (\psi_1 \diamond \mu) \lor \ldots \lor (\psi_n \diamond \mu) \quad (5.14)$$
\( \psi \) is satisfiable so there exists at least one \( \psi_i \) which is satisfiable. For such \( \psi_i \), we know that \( \psi_i \otimes \mu \) is satisfiable from the proof of (U3) in Lemma 5.6. Therefore, at least one disjunct of (5.14) is satisfiable and \( \psi \otimes \mu \) is satisfiable.

\((U4)\) If \( \psi_1 \leftrightarrow \psi_2 \) and \( \mu_1 \leftrightarrow \mu_2 \) then \( \psi_1 \otimes \mu_1 \leftrightarrow \psi_2 \otimes \mu_2 \).

Assume \( \psi_1 \leftrightarrow \psi_2 \) and \( \mu_1 \leftrightarrow \mu_2 \).

Hence \( \text{Mod}(\psi_1) = \text{Mod}(\psi_2) \) so we have:

\[
\psi_{11} \lor \ldots \lor \psi_{1n} \leftrightarrow \psi_{21} \lor \ldots \lor \psi_{2n} \tag{5.15}
\]

WLOG, we can say that \( \psi_{1i} \leftrightarrow \psi_{2i} \) for all \( i \).

Then, by using the proof of (U4) in Lemma 5.6, it follows that \( \psi_{1i} \otimes \mu_1 \leftrightarrow \psi_{2i} \otimes \mu_2 \) for all \( i \).

In full, then:

\[
(\psi_{11} \leftrightarrow \psi_{21}) \land \ldots \land (\psi_{1n} \leftrightarrow \psi_{2n}) \tag{5.16}
\]

\[
\Rightarrow (\psi_{11} \otimes \mu_1 \leftrightarrow \psi_{21} \otimes \mu_2) \land \ldots \land (\psi_{1n} \otimes \mu_1 \leftrightarrow \psi_{2n} \otimes \mu_2) \tag{5.17}
\]

\[
\Rightarrow (\psi_{11} \otimes \mu_1) \lor \ldots \lor (\psi_{1n} \otimes \mu_1) \leftrightarrow (\psi_{21} \otimes \mu_2) \lor \ldots \lor (\psi_{2n} \otimes \mu_2) \tag{5.18}
\]

\[
= (\psi_{11} \lor \ldots \lor \psi_{1n}) \otimes \mu_1 \leftrightarrow (\psi_{21} \lor \ldots \lor \psi_{2n}) \otimes \mu_2 \tag{5.19}
\]

\((U5)\) \((\psi \otimes \mu) \land \phi \) implies \( \psi \circ (\mu \land \phi) \).

\[
(\psi \circ \mu) \land \phi = ((\psi_1 \lor \ldots \lor \psi_n) \circ \mu) \land \phi \tag{5.20}
\]

\[
\overset{\text{def}}{=} ((\psi_1 \circ \mu) \lor \ldots \lor (\psi_n \circ \mu)) \land \phi \tag{5.21}
\]

\[
= ((\psi_1 \circ \mu) \land \phi) \lor \ldots \lor ((\psi_n \circ \mu) \land \phi) \tag{5.22}
\]

From the proof of (U5) in Lemma 5.6, we know that, for each \( \psi_i \), \((\psi_i \circ \mu) \land \phi \rightarrow \psi_i \circ (\mu \land \phi) \). So (5.22) implies:

\[
(\psi_1 \circ (\mu \land \phi)) \lor \ldots \lor (\psi_n \circ (\mu \land \phi)) \tag{5.23}
\]

\[
\overset{\text{def}}{=} (\psi_1 \lor \ldots \lor \psi_n) \circ (\mu \land \phi) \tag{5.24}
\]

\[
= \psi \circ (\mu \land \phi) \tag{5.25}
\]
(U6) If $\psi \circ \mu_1$ implies $\mu_2$ and $\psi \circ \mu_2$ implies $\mu_1$ then $\psi \circ \mu_1 \iff \psi \circ \mu_2$.

Assume $\psi \circ \mu_1 \rightarrow \mu_2$ and $\psi \circ \mu_2 \rightarrow \mu_1$

Therefore, $(\psi_1 \lor \ldots \lor \psi_n) \circ \mu_1 \rightarrow \mu_2$

Moreover, $(\psi_1 \circ \mu_1 \rightarrow \mu_2) \land \ldots \land (\psi_n \circ \mu_1 \rightarrow \mu_2)$

WRL to conclude that $(\psi_1 \circ \mu_2 \rightarrow \mu_1) \land \ldots \land (\psi_n \circ \mu_2 \rightarrow \mu_1)$.

By drawing on the proof of (U6) in Lemma 5.6, we can conclude that $\psi_1 \circ \mu_1 \iff \psi_i \circ \mu_2$

for all $i$. In full, then:

\[
(\psi_1 \circ \mu_1 \iff \psi_1 \circ \mu_2) \land \ldots \land (\psi_n \circ \mu_1 \iff \psi_n \circ \mu_2) \tag{5.26}
\]

\[
\Rightarrow (\psi_1 \circ \mu_1) \lor \ldots \lor (\psi_n \circ \mu_1) \iff (\psi_1 \circ \mu_2) \lor \ldots \lor (\psi_n \circ \mu_2) \tag{5.27}
\]

\[
= (\psi_1 \circ \mu_1 \rightarrow \mu_1) \leftrightarrow (\psi_1 \circ \mu_2) \rightarrow \mu_2
\]

\[
= \psi \circ \mu_1 \iff \psi \circ \mu_2 \tag{5.29}
\]

(U7) If $\psi$ is complete then $(\psi \circ \mu_1) \land (\psi \circ \mu_2)$ implies $\psi \circ (\mu_1 \lor \mu_2)$.

Here, we can refer directly to the proof of (U7) in Lemma 5.6 in which $\psi$ was assumed to be complete.

(U8) $(\psi_1 \lor \psi_2) \circ \mu \leftrightarrow (\psi_1 \circ \mu) \lor (\psi_2 \circ \mu)$.

For incomplete $\psi$, we transform $\psi$ into a disjunction of complete formulae and apply (U8) (see Definition 5.9). Therefore, by definition, $\circ$ satisfies (U8).

5.5 Minimality in Terms of Programs and Features.

Ostensibly, on feature integration, we are minimising change to the behaviour of the program whilst ensuring that the behaviour specified by the feature is imposed upon it. But what exactly do we mean by ‘minimising change to the behaviour of the program’?

Think of the denotation of an SMV program $P$, $[P]$, as a set of models, of which there is one for each possible state in the program. For example, if the program consists of two variables, $x$ and $y$, such that $x \in \{a, b, c\}$ and $y \in \{d, e, f, g\}$ then there will be $|\{a, b, c\}| \times |\{d, e, f, g\}| = 12$ states of the program and therefore, 12 models of $[P]$. Each
model $m$ specifies possible values for the variables in the next state; in other words, it specifies the states to which the program can transition from the current state (recall that $m$ represents a current state because there is one model for each state). For a feature $F$, $\langle F \rangle$ corresponds to a set of models as well. As far as the update operation is concerned, it goes through each model $m$ of $\langle P \rangle$. If $F$ specifies different next states for $m$, then $P$'s behaviour is changed. If $F$ does not alter the next states for $m$, then $P$'s behaviour remains unchanged. So for the updated program, $P'$, $\langle P' \rangle$ still corresponds to 12 models, and $F$ will have changed as few of the original 12 models as possible.

Note that there are not several possible degrees of change for a state; the nature of our $\circ$ operator means that either the feature changes a state, or it does not. So this change is minimal in that it changes the possible transitions for as few current states as possible.

Now we move on briefly, to consider orderings on features, in order that we can identify a feature which is minimal with respect to a particular program. Imagine we have two different features, $F_1$ and $F_2$, both of which introduce a particular desirable property to the program into which it is integrated. We want to find out whether $P + F_1$ or $P + F_2$ is minimal with respect to $P$ itself. Now this depends on our concept of minimality again. In reality, the choice of feature might depend on cost, efficiency or time. However, we can also order programs and features in terms of our ordering on models. We call the three programs $P$, $P_1$ and $P_2$, respectively, and we refer to the models of $\langle P \rangle$, $\langle P_1 \rangle$ and $\langle P_2 \rangle$.

Example 5.3 below shows one way in which we might order $P_1$ and $P_2$ with respect to $P$.

**Example 5.3** Assume that $P$, $P_1$ and $P_2$ all contain the same variables, in which case, they all have the same number of states, and therefore $|\text{Mod}(\langle P \rangle)| = |\text{Mod}(\langle P_1 \rangle)| = |\text{Mod}(\langle P_2 \rangle)|$.

Now we can compare the three programs by examining each state (and its corresponding model) in turn. There are various ways in which we could order the programs in relation
to each other. For example:

\[ P_1 \preceq_P P_2 \iff \exists m \cdot |d(m_{p_1}, m_p)| \leq |d(m_{p_2}, m_p)| \]
\[ \text{or } \exists m \cdot |d(m_{p_1}, m_p)| \leq |d(m_{p_2}, m_p)| \]
\[ \text{or } \forall m \cdot d(m_{p_1}, m_p) \subseteq d(m_{p_2}, m_p) \]
\[ \text{or } \exists m \cdot d(m_{p_1}, m_p) \subseteq d(m_{p_2}, m_p) \]

where the notion of distance between two models \( x \) and \( y \), \( d(x, y) \), was defined in Theorem 3.1, and \( m_{p_1}, m_{p_2} \) represent corresponding models of \( [P_1] \) and \( [P_2] \) respectively. Corresponding models represent next values for the same current state, which is referred to, in general terms, as \( m \) in the equation above.

If the programs vary in the number of variables, then there are many more possible ways in which they might be ordered in relation to each other. For example, the variables themselves might be compared with each other prior to carrying out a comparison as in Example 5.3.

It is clear, then, that this model-theoretic view of programs and features provides us with a useful quantitative tool, which enables us to differentiate between them in numerous different ways. This is a practical topic which could be explored in much greater depth in future.
Chapter 6

Conclusions and Directions for Future Work

In this thesis, we have formalised a relationship between the two previously unrelated areas of feature integration and update. The nonmonotonic nature of the two subjects had been recognised before, but never investigated in depth.

We took Plath and Ryan’s SMV feature construct and, in Chapter 4, we presented a method for translating SMV programs and features into propositional logic via an abstract representation. The temporal nature of SMV was captured by using two types of propositions: the set $X$ for ‘current’ values of variables, and the set $X'$ for ‘next’ values of variables. We also explained why a more ‘natural’ denotation was unsuitable and gave an equivalent denotation. Then we reformulated the eight rationality postulates for update in terms of programs and features and proved that they held in that context, thus providing evidence of a correlation between the two areas.

However, this result was not entirely satisfactory because it did not enable us to make use of Katsuno and Mendelzon’s representation theorem for update. We had proved that the update axioms held in the special case when formulae represented programs and features, but the representation theorem is only applicable in the general case.

Therefore, in Chapter 5, we formulated a new update operator on arbitrary formulae which, crucially, defines SMV feature integration when the formulae represent programs and features. In other words, it is an operator which specialises to feature integration.
Conclusions and Directions for Future Work

We formally proved that the new update operator does indeed perform SMV feature integration and that it satisfies the update axioms. Finally, we discussed the representation theorem in terms of programs and features, although in-depth research into this area is left to future work.

We set out to explore the relationship between two subjects which are inherently nonmonotonic in nature, and we achieved results which exceeded our expectations. In particular, this was true of the result presented in Chapter 5, where we formulated the new update operator. This result enables us to make use of the semantic machinery associated with update.

The main contribution of this work is that it gives a theoretical underpinning to SMV feature integration, an operation which is widely used and cited in feature interaction research. The precise way in which the operation modifies the base software system has never been fully understood in theoretical terms. This is unsatisfactory: an operation which alters the properties of a system is invasive. The more we know about the operation's underlying properties, the better equipped we are to develop it in an intelligent and useful way. Furthermore, the work in this thesis has formulated a relationship between feature integration and the logic of theory change, which has benefited from twenty years of research and is a very well-understood, widely-accepted theoretical concept.

A detailed study of minimality in terms of programs and features would constitute another substantial piece of work, which is beyond the scope of this thesis, but it is a vital avenue for future work. If this work is carried out, it should be possible to find a means by which features can be ordered in terms of their closeness to the base system. In turn, this work could yield a hierarchical classification of features in terms of how disruptive they are to the base system.

Another possibility is to consider other practical methods of feature integration from the literature and investigate how they relate to the update operation. In particular, Gammelgaard and Kristensen's logical approach [GK94] and Calder and Miller's SPIN feature construct [CM01] are likely candidates. It should be possible to reuse our denotation based on the sets of atomic propositions, $X$ and $X'$, and it would be interesting to find out whether our new update operator would further generalise to these other feature
integration techniques.

We could continue to work on the SMV feature construct. We have not considered the TREAT clauses of the feature construct. In addition, there are two further types of clause specified in the Mocha feature construct by Cassez, Ryan and Schobbens [CRS01] which may well also fit into the update paradigm.

We speculate that TREAT clauses could be translated into this framework in a similar way to IMPOSE clauses, although the process is a little more complicated because the TREAT construct changes the read-value of variables as opposed to the write-value, and the propositional logic translation could not be achieved as naturally as it was for IMPOSE.

We also decided not to consider the initialisation values of SMV variables in this work because we wanted to keep the process as simple as possible. However, because the initialisation values remain constant throughout a program, incorporating them into a propositional logic denotation would be trivial. In order to encode it, we would simply need to add some initialisation propositions to the denotation.

Finally, we indicated some possible avenues for future work in Chapter 3. Feature integration could be considered with respect to screened revision, iterated revision and local change as they are all operations of theory change, aspects of which appear to be relevant to feature integration.

Thus, the work in this thesis leads on to several other related research topics. It is hoped that the update operation will prove to be a useful theoretical tool for the software engineering community and that, in turn, feature integration will be a valuable new application area for the logic of theory change.
Appendix A

Hand-Coded Examples of Update Using the Results in this Thesis

In this appendix, we return to the intruder deterrent light example of Chapter 2. We give the propositional formulation of this example and show exactly how it works with the ∈ operator introduced in Section 5.2. The idea here is not to present new results but simply to demonstrate to the interested reader how the main results of this thesis may be applied to a small-scale example in practice.

First, we introduce a night_time feature which can be integrated into the basic intruder deterrent light system - see Example 2.4. It is presented below in both its SMV and propositional logic representations. We will then go on to compare how the two possible features, night_time and evening_hours, disrupt the basic system from a model-theoretic point of view and briefly discuss how the two possible featured programs might be ordered with respect to the base system.

**Example A.1** We introduce a feature to be integrated into the program in Example 2.4 which ensures that the intruder deterrent light is off during the night. night_time is a boolean variable which is true during the night.

IF night_time THEN IMPOSE next(switch) := off;

**Example A.2** The program of Example 2.4 into which the feature of Example A.1 has been integrated.
MODULE main
VAR
    switch : {on, off};
    night_time : boolean;
ASSIGN
    init(switch) := off;
    next(switch) := case
        night_time : off;
        1 : case
            switch = on : off;
            1 : {on, off}
        esac;
    esac;
SPEC AG(switch = on -> AF switch = off)

Example A.3 The denotation of the night_time feature.

\[ \text{night_time} \rightarrow \text{switch}'_{\text{off}} \land \neg \text{switch}'_{\text{on}} \]

Example A.4 The denotation of the basic intruder deterrent light program into which the night_time feature has been integrated.

\[
(night\_time \rightarrow \text{switch}'_{\text{off}} \land \neg \text{switch}'_{\text{on}}) \land \\
(\neg night\_time \land (\text{switch} = \text{on}) \rightarrow \text{switch}'_{\text{off}} \land \neg \text{switch}'_{\text{on}}) \land \\
(\neg night\_time \land \neg (\text{switch} = \text{on}) \rightarrow \text{switch}'_{\text{on}} \land \text{switch}'_{\text{off}}) \land \\
(night\_time'_{T} \land night\_time'_{F})
\]

Figure A.1 shows a state transition diagram for the intruder deterrent light which consists of eight states (for there are three boolean variables). The black transitions are those of the base system, the blue are those of the \textit{evening_hours} feature, and the red are those of the \textit{night_time} feature. Crosses on a transition denote deletion by the corresponding feature. In the diagram, we abbreviate \textit{evening_hours} to e\_h and \textit{night_time} to n\_t.
Figure A.1: State transition diagram for the intruder deterrent light example.
The equivalent denotation of SMV which was presented at the end of Chapter 5 means that a program may be represented by a disjunction of conjunctions, each of which is complete over \( \textit{Vars} \). Therefore, each disjunct of the formula represents a model. Example A.5 shows several tables which give all the models for a particular SMV program. Each numbered row represents a disjunct of the program’s formula, or a model. Propositions which are modified by the features are shown in blue for the \textit{evening hours} feature and in red for the \textit{night time} feature.

\textbf{Example A.5} The Intruder Deterrent Light from a Model-Theoretic Point of View.

Let the basic intruder deterrent light program be \( P \), the \textit{evening hours} feature be \( F_1 \), and the \textit{night time} feature be \( F_2 \).

In this example, we show \( \text{Mod}(\llbracket P \rrbracket) \), \( \text{Mod}(\llbracket P + F_1 \rrbracket) \) and \( \text{Mod}(\llbracket P + F_2 \rrbracket) \) followed by calculations of \( \text{Mod}(\llbracket P \rrbracket \diamond \llbracket F_1 \rrbracket) \) and \( \text{Mod}(\llbracket P \rrbracket \diamond \llbracket F_2 \rrbracket) \). Each numbered row in each table corresponds to a model of the associated formula.

\begin{center}
\begin{tabular}{llllllllll}
\hline
\textbf{Mod}(\llbracket P \rrbracket) \\
1. & \( s = \text{on} \) & \( e \text{\_light} \) & \( n \downarrow \) & \( \neg s\text{'on} \) & \( s\text{'off} \) & \( e_h \text{\_light} \uparrow \) & \( e_h \text{'light} \downarrow \) & \( n \downarrow_T \) & \( n \downarrow_F \) \\
2. & \( s = \text{off} \) & \( e \text{\_light} \) & \( n \downarrow \) & \( s\text{'on} \) & \( s\text{'off} \) & \( e_h \text{\_light} \downarrow \) & \( e_h \text{'light} \uparrow \) & \( n \downarrow_T \) & \( n \downarrow_F \) \\
3. & \( s = \text{on} \) & \( \neg e \text{\_light} \) & \( n \downarrow \) & \( \neg s\text{'on} \) & \( s\text{'off} \) & \( e_h \text{\_light} \uparrow \) & \( e_h \text{'light} \downarrow \) & \( n \downarrow_T \) & \( n \downarrow_F \) \\
4. & \( s = \text{on} \) & \( e \text{\_light} \) & \( \neg n \downarrow \) & \( \neg s\text{'on} \) & \( s\text{'off} \) & \( e_h \text{\_light} \downarrow \) & \( e_h \text{'light} \uparrow \) & \( n \downarrow_T \) & \( n \downarrow_F \) \\
5. & \( s = \text{off} \) & \( \neg e \text{\_light} \) & \( n \downarrow \) & \( s\text{'on} \) & \( s\text{'off} \) & \( e_h \text{\_light} \downarrow \) & \( e_h \text{'light} \uparrow \) & \( n \downarrow_T \) & \( n \downarrow_F \) \\
6. & \( s = \text{off} \) & \( e \text{\_light} \) & \( \neg n \downarrow \) & \( s\text{'on} \) & \( s\text{'off} \) & \( e_h \text{\_light} \downarrow \) & \( e_h \text{'light} \uparrow \) & \( n \downarrow_T \) & \( n \downarrow_F \) \\
7. & \( s = \text{on} \) & \( \neg e \text{\_light} \) & \( \neg n \downarrow \) & \( \neg s\text{'on} \) & \( s\text{'off} \) & \( e_h \text{\_light} \uparrow \) & \( e_h \text{'light} \downarrow \) & \( n \downarrow_T \) & \( n \downarrow_F \) \\
8. & \( s = \text{off} \) & \( \neg e \text{\_light} \) & \( \neg n \downarrow \) & \( s\text{'on} \) & \( s\text{'off} \) & \( e_h \text{\_light} \downarrow \) & \( e_h \text{'light} \uparrow \) & \( n \downarrow_T \) & \( n \downarrow_F \) \\
\hline
\end{tabular}
\end{center}
\[ Mod(\|P + F1\|) \]

1. \( s = on \) \( e_J \) \( n \downarrow \) \( s'_{on} \) \( \neg s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
2. \( s = off \) \( e_J \) \( n \downarrow \) \( s'_{on} \) \( \neg s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
3. \( s = on \) \( \neg e_J \) \( n \downarrow \) \( s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
4. \( s = on \) \( e_J \) \( \neg n \downarrow \) \( s'_{on} \) \( \neg s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
5. \( s = off \) \( \neg e_J \) \( n \downarrow \) \( s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
6. \( s = off \) \( e_J \) \( \neg n \downarrow \) \( s'_{on} \) \( \neg s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
7. \( s = on \) \( \neg e_J \) \( \neg n \downarrow \) \( \neg s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
8. \( s = off \) \( \neg e_J \) \( \neg n \downarrow \) \( \neg s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)

\[ Mod(\|P + F2\|) \]

1. \( s = on \) \( e_J \) \( n \downarrow \) \( \neg s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
2. \( s = off \) \( e_J \) \( n \downarrow \) \( \neg s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
3. \( s = on \) \( \neg e_J \) \( n \downarrow \) \( \neg s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
4. \( s = on \) \( e_J \) \( \neg n \downarrow \) \( \neg s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
5. \( s = off \) \( \neg e_J \) \( n \downarrow \) \( \neg s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
6. \( s = off \) \( e_J \) \( \neg n \downarrow \) \( s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
7. \( s = on \) \( \neg e_J \) \( \neg n \downarrow \) \( \neg s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
8. \( s = off \) \( \neg e_J \) \( \neg n \downarrow \) \( s'_{on} \) \( s'_{off} \) \( e_{J_T} \) \( e_{J_F} \) \( n_{J_T} \) \( n_{J_F} \)
Hand-Coded Examples of Update Using the Results in this Thesis

\[ \text{Mod}(\mathbb{P} \circ \mathbb{F}1) \]

1. \( \text{Mod}(s = \text{on} \land e_j \land n_j \land \neg s'_n \land s'_o \land e_{\mathbb{J}'_T} \land e_{\mathbb{J}'_F} \land n_{\mathbb{J}'_T} \land n_{\mathbb{J}'_F}) \)
   \[ \triangleq (s = \text{on} \land e_j \land n_j \land (e_j \land s'_o \land \neg s'_o \land \neg e_j)) \]

2. \( \text{Mod}(s = \text{off} \land e_j \land n_j \land s'_n \land s'_o \land e_{\mathbb{J}'_T} \land e_{\mathbb{J}'_F} \land n_{\mathbb{J}'_T} \land n_{\mathbb{J}'_F}) \)
   \[ \triangleq (s = \text{off} \land e_j \land n_j \land (e_j \land s'_o \land \neg s'_o \land \neg e_j)) \]

3. \( \text{Mod}(s = \text{on} \land \neg e_j \land n_j \land \neg s'_n \land s'_o \land e_{\mathbb{J}'_T} \land e_{\mathbb{J}'_F} \land n_{\mathbb{J}'_T} \land n_{\mathbb{J}'_F}) \)
   \[ \triangleq (s = \text{on} \land \neg e_j \land n_j \land (e_j \land s'_o \land \neg s'_o \land \neg e_j)) \]

4. \( \text{Mod}(s = \text{on} \land e_j \land \neg n_j \land \neg s'_n \land s'_o \land e_{\mathbb{J}'_T} \land e_{\mathbb{J}'_F} \land n_{\mathbb{J}'_T} \land n_{\mathbb{J}'_F}) \)
   \[ \triangleq (s = \text{on} \land \neg e_j \land n_j \land (e_j \land s'_o \land \neg s'_o \land \neg e_j)) \]

5. \( \text{Mod}(s = \text{off} \land \neg e_j \land n_j \land s'_n \land s'_o \land e_{\mathbb{J}'_T} \land e_{\mathbb{J}'_F} \land n_{\mathbb{J}'_T} \land n_{\mathbb{J}'_F}) \)
   \[ \triangleq (s = \text{off} \land \neg e_j \land n_j \land (e_j \land s'_o \land \neg s'_o \land \neg e_j)) \]

6. \( \text{Mod}(s = \text{off} \land e_j \land \neg n_j \land s'_n \land s'_o \land e_{\mathbb{J}'_T} \land e_{\mathbb{J}'_F} \land n_{\mathbb{J}'_T} \land n_{\mathbb{J}'_F}) \)
   \[ \triangleq (s = \text{off} \land \neg e_j \land n_j \land (e_j \land s'_o \land \neg s'_o \land \neg e_j)) \]

7. \( \text{Mod}(s = \text{on} \land \neg e_j \land \neg n_j \land \neg s'_n \land s'_o \land e_{\mathbb{J}'_T} \land e_{\mathbb{J}'_F} \land n_{\mathbb{J}'_T} \land n_{\mathbb{J}'_F}) \)
   \[ \triangleq (s = \text{on} \land \neg e_j \land \neg n_j \land (e_j \land s'_o \land \neg s'_o \land \neg e_j)) \]

8. \( \text{Mod}(s = \text{off} \land \neg e_j \land \neg n_j \land s'_n \land s'_o \land e_{\mathbb{J}'_T} \land e_{\mathbb{J}'_F} \land n_{\mathbb{J}'_T} \land n_{\mathbb{J}'_F}) \)
   \[ \triangleq (s = \text{off} \land \neg e_j \land \neg n_j \land (e_j \land s'_o \land \neg s'_o \land \neg e_j)) \]
Hand-Coded Examples of Update Using the Results in this Thesis

104

\[ \text{Mod}(\|P\| \cup \|F2\|) \]

1. \( \text{Mod}(s = on \land e_J \land n_J \land -s'_on \land s'_off \land e_J'T \land e_J'F \land n_J'T \land n_J'F) \)
   \( \bullet (s = on \land e_J \land n_J \land (n_J \land -s'_on \land s'_off) \vee -n_J) \)

2. \( \text{Mod}(s = off \land e_J \land n_J \land s'_on \land s'_off \land e_J'T \land e_J'F \land n_J'T \land n_J'F) \)
   \( \bullet (s = off \land e_J \land n_J \land (n_J \land -s'_on \land s'_off) \vee -n_J) \)

3. \( \text{Mod}(s = on \land -e_J \land n_J \land -s'_on \land s'_off \land e_J'T \land e_J'F \land n_J'T \land n_J'F) \)
   \( \bullet (s = on \land -e_J \land n_J \land (n_J \land -s'_on \land s'_off) \vee -n_J) \)

4. \( \text{Mod}(s = on \land e_J \land n_J \land -s'_on \land s'_off \land e_J'T \land e_J'F \land n_J'T \land n_J'F) \)
   \( \bullet (s = on \land e_J \land n_J \land (n_J \land -s'_on \land s'_off) \vee -n_J) \)

5. \( \text{Mod}(s = off \land -e_J \land n_J \land s'_on \land s'_off \land e_J'T \land e_J'F \land n_J'T \land n_J'F) \)
   \( \bullet (s = off \land -e_J \land n_J \land (n_J \land -s'_on \land s'_off) \vee -n_J) \)

6. \( \text{Mod}(s = off \land e_J \land n_J \land -s'_on \land s'_off \land e_J'T \land e_J'F \land n_J'T \land n_J'F) \)
   \( \bullet (s = off \land e_J \land n_J \land (n_J \land -s'_on \land s'_off) \vee -n_J) \)

7. \( \text{Mod}(s = on \land -e_J \land n_J \land -s'_on \land s'_off \land e_J'T \land e_J'F \land n_J'T \land n_J'F) \)
   \( \bullet (s = on \land -e_J \land n_J \land (n_J \land -s'_on \land s'_off) \vee -n_J) \)

8. \( \text{Mod}(s = off \land -e_J \land n_J \land -s'_on \land s'_off \land e_J'T \land e_J'F \land n_J'T \land n_J'F) \)
   \( \bullet (s = off \land -e_J \land n_J \land (n_J \land -s'_on \land s'_off) \vee -n_J) \)

\[
\begin{align*}
1. & s = on \quad e_J \quad n_J \quad -s'_on \quad s'_off \quad e_J'T \quad e_J'F \quad n_J'T \quad n_J'F \\
2. & s = off \quad e_J \quad n_J \quad -s'_on \quad s'_off \quad e_J'T \quad e_J'F \quad n_J'T \quad n_J'F \\
3. & s = on \quad -e_J \quad n_J \quad -s'_on \quad s'_off \quad e_J'T \quad e_J'F \quad n_J'T \quad n_J'F \\
4. & s = on \quad e_J \quad -n_J \quad -s'_on \quad s'_off \quad e_J'T \quad e_J'F \quad n_J'T \quad n_J'F \\
5. & s = off \quad -e_J \quad n_J \quad -s'_on \quad s'_off \quad e_J'T \quad e_J'F \quad n_J'T \quad n_J'F \\
6. & s = off \quad e_J \quad -n_J \quad -s'_on \quad s'_off \quad e_J'T \quad e_J'F \quad n_J'T \quad n_J'F \\
7. & s = on \quad -e_J \quad -n_J \quad -s'_on \quad s'_off \quad e_J'T \quad e_J'F \quad n_J'T \quad n_J'F \\
8. & s = off \quad -e_J \quad -n_J \quad -s'_on \quad s'_off \quad e_J'T \quad e_J'F \quad n_J'T \quad n_J'F
\end{align*}
\]

Note that, as shown in Theorem 5.1, \( \text{Mod}(\|P + F1\|) \Leftrightarrow \text{Mod}(\|P\| \cup \|F1\|) \) and \( \text{Mod}(\|P + F2\|) \Leftrightarrow \text{Mod}(\|P\| \cup \|F2\|) \).

Now we briefly refer to the discussion at the end of Chapter 5 because, with two
features $F1$ and $F2$, we can consider how the corresponding featured programs, $P1$ and $P2$, may be ordered with respect to the base program $P$.

Unfortunately, if we use the possible ordering examples given in Example 5.3, in the case of the $\forall m$ examples, $P1$ and $P2$ are incomparable with respect to $P$ because neither of the examples holds (although note that $\forall m . |d(m_{P2}, m_P)| \leq |d(m_{P1}, m_P)|$ would hold if it were not for model 5), and with the $\exists m$ examples, they both hold for both orderings, i.e. $P1 \leq_P P2$ and $P2 \leq_P P1$ (again, thanks to model 5) so in this case, $P1$ and $P2$ are equivalent with respect to $P$.

The only way in which we can formulate an ordering which does not render the programs either incomparable or equivalent is if we crudely compare the total number of propositional values which are modified by the features, as in Example A.6 below:

**Example A.6** $P1 \leq_P P2 \iff \Sigma_{\forall m} |d(m_{P1}, m_P)| \leq \Sigma_{\forall m} |d(m_{P2}, m_P)|$

If we use the crude ordering given in Example A.6, then we get that $P2 \leq_P P1$ because, to put it simply, $P2$ modifies less of $P$’s propositional values than $P1$ does. However, this is a weak means of comparison and, as we have stated previously, in practice there will be other higher level factors to take into account when we compare two features, such as time, cost and efficiency, so we leave further consideration of this topic to future research.
Appendix B

A Photograph

Here is a diagram that I drew on the blackboard during a talk at a Dagstuhl seminar in February 2003. It was amusingly modified by an academic vandal who has yet to be identified!
Bibliography


Bibliography


Bibliography


