Feature Integration as an Operation of Theory Change

Hannah Harris [1] and **Mark Ryan** [1]

Abstract. We formalise a relationship between two previously unconnected areas: theory change and feature interaction. This will provide 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.

Update is an operation of theory change which is closely related to belief revision. The principal difference lies in the fact that belief revision models changing beliefs about a static world whereas update models a changing world.

A feature is a unit of functionality which extends or modifies the behaviour of the system into which it is integrated. The feature interaction problem arises when two or more features interact, causing the system to exhibit unexpected, and often undesirable, behaviour. Many approaches to feature integration and interaction detection have been proposed. In this research we use a feature construct for the model checker SMV.

We show that there is a strong connection between update and feature integration by, preliminarily, formulating SMV and the feature construct in propositional logic. We then go on to prove that the eight rationality postulates for update hold in the context of this theoretical formulation.

1 INTRODUCTION

A feature is a unit of functionality which extends or modifies a system by overriding its behaviour in some way. Feature integration is a nonmonotonic operation because it has the potential to disrupt properties of the system which were known to hold previously. In Example 1, we consider a very simple expert system which is used to diagnose certain illnesses based on their symptoms.

Example 1 *Base system (to diagnose a cold):*

- **if** *blocked nose* **then** *patient has a cold*

Now we may add two features (to diagnose flu and tonsillitis):

- **if** *fever and headache* **then** *patient has flu*
- **if** *fever and headache and sore throat* **then** *patient has tonsillitis*

Each of the features may be integrated successfully into the base system. However, if integrated together, they will interact undesirably, assuming the rules appear in the order in which they appear above. If a patient has a fever, a headache, and a sore throat, the system above will diagnose flu, when it should diagnose tonsillitis. This is because one feature blocks the other. A resolution strategy needs to be implemented in order to determine the correct priority of rule application.

[1] University of Birmingham, Birmingham, B15 2TT, UK.
http://www.cs.bham.ac.uk/~{heh,mdr}

Example 1 shows how two features can interact due to the inherently nonmonotonic nature of feature integration. Feature interaction is a recognised phenomenon which is manifested by a system whose functionality may be extended or modified by two or more features. Interactions occur when such features cause the system to exhibit unexpected, and often undesirable, behaviour which does not form part of the specification of either the system or one of the features. A classic example is of a telephone system.

Example 2 *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. If B subscribes to CW (Call Waiting) and A calls B when 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.*

It is an accepted fact that several issues of nonmonotonicity arise in the context of feature interaction [11]. However, no formal connection between feature integration and nonmonotonic reasoning has been established.

In our research we have investigated the relationship between feature integration and theory change, the latter being an area of research whose relationship with nonmonotonic reasoning is well established [7]. 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. Our motivation for 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.

Although we have identified this parallel between theory change and feature integration, there are two principal factors which render it non-trivial: (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) a feature is often specified in a language which is different from that used to specify the system, whereas in theory change, new knowledge is normally in the same type of language as that which the knowledge base itself

contains.

In this paper, 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. Specifically, we give background material on the theory change operation of update (Section 2) and a feature constuct for the model-checker SMV [10] (Section 3). Then in Section 4, we go on to show our theoretical formulation of SMV and the feature construct, and that the eight rationality postulates for update hold for this theoretical formulation. We give our conclusions and indicate possible future work in Section 5.

2 THE UPDATE OPERATION

Update is an operation which is closely related to belief revision. Katsuno and Mendelzon propose eight rationality postulates, (U1)∽(U8), for update in [5]. 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. The eight axioms are given below. $\psi \diamond \mu$ denotes the result of updating a formula ψ by a formula μ.

(U1) $\psi \diamond \mu$ implies μ.
(U2) If ψ implies μ then $\psi \diamond \mu$ is equivalent to ψ.
(U3) If both ψ and μ are satisfiable then $\psi \diamond \mu$ is also satisfiable.
(U4) If $\psi_1 \leftrightarrow \psi_2$ and $\mu_1 \leftrightarrow \mu_2$ then $\psi_1 \diamond \mu_1 \leftrightarrow \psi_2 \diamond \mu_2$.
(U5) $(\psi \diamond \mu) \wedge \phi$ implies $\psi \diamond (\mu \wedge \phi)$.
(U6) If $\psi \diamond \mu_1$ implies μ_2 and $\psi \diamond \mu_2$ implies μ_1 then $\psi \diamond \mu_1 \leftrightarrow \psi \diamond \mu_2$.
(U7) If ψ is complete then $(\psi \diamond \mu_1) \wedge (\psi \diamond \mu_2)$ implies $\psi \diamond (\mu_1 \vee \mu_2)$.
(U8) $(\psi_1 \vee \psi_2) \diamond \mu \leftrightarrow (\psi_1 \diamond \mu) \vee (\psi_2 \diamond \mu)$.

There are several salient differences between update and revision apart from the general one given above. These differences are explored in detail in [5], but we summarise them here. First of all, let us take a model-theoretic point of view, where ψ is the knowledge base and μ is the sentence representing the new information. Revision selects the models of μ which are closest to the set of models of ψ. Update selects the set of closest models of μ for each model of ψ 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 ψ 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 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 ψ 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 μ which are closest to each of them.

Example 3 *(based on an example in [2]). Consider a situation in which either a book or a magazine is on a table: $b \vee m$. 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 $\neg b \wedge m$, 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 $b \vee m$ will still be true in the real world.

3 FEATURE INTERACTION AND THE SMV FEATURE CONSTRUCT

In this section, we discuss the feature interaction problem in more detail (Section 3.1). We then go on to give a brief introduction to the SMV model checker in Section 3.2. Lastly, we present SFI (SMV Feature Integrator), the SMV feature construct, in Section 3.3.

3.1 The Feature Interaction Problem

The feature interaction problem is ubiquitous in telecommunications and software systems. Instances of its manifestation can be found in many different areas, from intelligent telephone systems (of which examples may be found throughout the feature interaction literature, see [1, 6, 3]) to lift systems, and more recently in email and IP telephony. 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. These methods are often classified under the broad categories of software engineering, formal, and pragmatic approaches.

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 [10]. 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.

3.2 Introduction to SMV

SMV (Symbolic Model Verifier) [9] 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 [8] and [9] and CTL is described in [4].

Example 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.*

```
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 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 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.

3.3 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 next(`*x*`) :=` *expr*`;`

means that assignments such as `next(`*x*`) :=` *oldexpr*`;` have *oldexpr* replaced by:

```
case
  cond : expr;
  1    : oldexpr;
esac
```

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

Example 5 *We introduce a feature to be integrated into the program in Example 4. We want the intruder deterrent light to be on constantly in the evening between the hours of 8pm and 11pm, say.* `evening_hours` *is a boolean variable which is true during these hours.*

```
IF evening_hours THEN
            IMPOSE next(switch) := on;
```

Example 6 *The program of Example 4 into which the feature of Example 5 has been integrated.*

```
MODULE main
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 AG(switch = on -> AF switch = off)
```

In the resulting program, the behaviour of the `evening_hours` variable is unspecified, so it will assume values nondeterministically. In this case, the CTL formula is not satisfied because if `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 `switch` has the value `on` forever.

A full explanation of the work summarised in this section can be found in [10].

4 FEATURE INTEGRATION IN SMV AS AN UPDATE OPERATION

In this section, we show that feature integration in SMV can be viewed as an update operation. In order to do this, we will prove that the eight rationality postulates for update, (U1)⌣(U8) (shown in Section 2), hold for feature integration by means of the SMV feature construct. Initially, then, it is necessary to give a propositional logic formulation of SMV and the feature construct because the eight postulates for update are defined in terms of propositional logic; we do this in Section 4.1. Then, in Section 4.2, we sketch our proofs of (U1)⌣(U8).

4.1 Theoretical Formulation

In order to represent SMV in propositional logic, we begin by giving an abstract representation of a program, P. Definition 1 shows how this may be done.

Definition 1 (Abstract Representation for an SMV Program, P)

$$P = \{x_1 : \{\phi_{11} \rightarrow A_{11}, \ldots, \phi_{1l_1} \rightarrow A_{1l_1}\}, \ldots, \\ x_n : \{\phi_{n1} \rightarrow A_{n1}, \ldots, \phi_{nl_n} \rightarrow A_{nl_n}\}\}$$

With each variable x_i is associated a set of case-value pairs which represent the case statements to evaluate the next value of x_i in terms of the variables' current values. In Definition 1 then, the ϕ-expressions are logical formulae over the variables of P; they represent the cases, which are covering and exclusive for each variable. Each A-expression represents a set of possible next values for x_i. We write V_P for the set $\{x_1, \ldots, x_n\}$ of variables occurring in P.

Example 7 *The abstract representation for the program of Example 4 is as follows:*

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

Note that, in the second clause of Example 7, the antecedent of the first clause, $switch = on$, has to be negated in order that the cases are exclusive. The reason for this 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.

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 α, B in features where we used ϕ, A in Definition 1. We write V_F for the set $\{x_1, \ldots, x_n\}$ of variables occurring in F.

Definition 2 (Abstract Representation for a Feature, F)

$$F = \{x_1 : \{\alpha_{11} \rightarrow B_{11}, \ldots, \alpha_{1m_1} \rightarrow B_{1m_1}\}, \ldots, \\ x_n : \{\alpha_{n1} \rightarrow B_{n1}, \ldots, \alpha_{nm_n} \rightarrow B_{nm_n}\}\}$$

Example 8 *The abstract representation for the feature of Example 5 is as follows:*

$$\{switch : \{evening\_hours \rightarrow \{on\}\}\}$$

In Definition 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 [10] and summarised in Section 3.3 of this paper. Note that $P + F$ is a program in itself so its abstract representation is equivalent to that of Definition 1. Also note that we assume, without loss of generality, that $V_F \subseteq V_P$; if a program does not explicitly specify the behaviour of a variable, its behaviour is assumed to be non-deterministic.

Definition 3 (Abstract Representation for a Featured Program, P + F) *Let P and F be defined as in Definitions 1 and 2 respectively. Since $V_F \setminus V_P = \emptyset$, $x \in V_F \cap V_P$ or $x \in V_P \setminus V_F$. $P + F$ is defined as follows:*
Suppose $x_i \in V_F \cap V_P$. Then:

$$(x_i : \{\alpha_{ij} \rightarrow B_{ij} \mid 1 \leq j \leq m_i\} \\ \cup \{\neg\alpha_i \wedge \phi_{ik} \rightarrow A_{ik} \mid 1 \leq k \leq l_i\}) \in (P + F)$$

where $\alpha_i = \bigvee_{j=1}^{m_i} \alpha_{ij}$.
Suppose $x_i \in V_P \setminus V_F$. Then:

$$(x_i : \{\phi_{ik} \rightarrow A_{ik} \mid 1 \leq k \leq l_i\}) \in (P + F)$$

Example 9 *The abstract representation for the featured program of Example 6 is as follows:*

$$\{switch : \{evening\_hours \rightarrow \{on\}, \\ \neg evening\_hours \wedge (switch = on) \rightarrow \{off\}, \\ \neg evening\_hours \wedge \neg(switch = on) \rightarrow \{on, off\}, \\ evening\_hours : \{1 \rightarrow \{0, 1\}\}$$

For the denotation $[\![P]\!]$ of P, we need some sets of atomic propositions.

$V_P = \{x_i \mid 1 \leq i \leq n$ where n is the number of variables in $P\}$
$N_{P,x_i} = \{x'_{i,v} \mid v \in \text{type}(x_i)\}$, for each $x_i \in V_P$.

type(x) denotes a finite set of possible values for x, e.g. if x is boolean, type(x) = {0,1}. V_P contains the propositions denoting the current values of variables. There is a set $N_{P,x}$ for each variable $x \in V_P$ which contains propositions relating to the possible next values for x according to its type.

Now we are ready to look at the formal denotation of P, F and $P + F$.

Definition 4 (Denotation of an SMV Program, P) *Let P be defined as in Definition 1. Then:*

$$[\![P]\!] = \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{l_i} (\phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} x'_{i,v} \wedge \bigwedge_{v \notin A_{ij}} \neg x'_{i,v})$$

where $x'_{i,v} \in N_{P,x_i}$.

Example 10 *The denotation of the abstract program of Example 7 is the following:*

$$((switch = on) \rightarrow switch'_{off} \wedge \neg switch'_{on}) \wedge \\ (\neg(switch = on) \rightarrow switch'_{on} \wedge switch'_{off})$$

Definition 5 (Denotation of a Feature, F) *Let F be defined as in Definition 2. Then:*

$$[\![F]\!] = \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m_i} (\alpha_{ij} \rightarrow \bigwedge_{v \in B_{ij}} x'_{i,v} \wedge \bigwedge_{v \notin B_{ij}} \neg x'_{i,v})$$

where $x'_{i,v} \in N_{F,x_i}$.

Example 11 *The denotation of the abstract feature of Example 8 is the following:*

$$evening\_hours \rightarrow switch'_{on} \wedge \neg switch'_{off}$$

Lemma 1 (Denotation of a Featured Program, P + F) *Let P and F be defined as in Definitions 1 and 2 respectively. Then:*

$$[\![P + F]\!] = \bigwedge_{i=1}^{n} (\bigwedge_{j=1}^{m_i} (\alpha_{ij} \rightarrow \bigwedge_{v \in B_{ij}} x'_{i,v} \wedge \bigwedge_{v \notin B_{ij}} \neg x'_{i,v}) \wedge \\ \bigwedge_{j=1}^{l_i} (\neg\alpha_i \wedge \phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} x'_{i,v} \wedge \bigwedge_{v \notin A_{ij}} \neg x'_{i,v})) \wedge \\ \bigwedge_{i=1}^{n} (\bigwedge_{j=1}^{l_i} (\phi_{ij} \rightarrow \bigwedge_{v \in A_{ij}} x'_{i,v} \wedge \bigwedge_{v \notin A_{ij}} \neg x'_{i,v}))$$

where $\alpha_i = \bigvee_{j=1}^{m_i} \alpha_{ij}$ and $x'_{i,v} \in N_{P+F,x_i}$.

Example 12 *The denotation of the abstract featured program of Example 9 is the following:*

$(evening\_hours \rightarrow switch'_{on} \wedge \neg switch'_{off}) \wedge$
$(\neg evening\_hours \wedge (switch = on) \rightarrow switch'_{off} \wedge \neg switch'_{on}) \wedge$
$(\neg evening\_hours \wedge \neg(switch = on) \rightarrow switch'_{on} \wedge switch'_{off}) \wedge$
$(evening\_hours'_0 \wedge evening\_hours'_1)$

4.2 Proving the Update Axioms

In this section, we prove that the eight rationality postulates for update are true of the theoretical formulation of feature integration as presented in Section 4.1. We give a definition and our theorem, and then in the proof section, we summarise the proofs for each postulate.

Definition 6 *We define the update operator on formulas which are denotations of programs and features. Let $\psi = [\![P]\!]$ and $\mu = [\![F]\!]$. Then $\psi \diamond \mu = [\![P + F]\!]$.*

Theorem. Let ψ, ψ_1, ψ_2 be denotations of programs, and μ, μ_1, μ_2, ϕ be denotations of features. Then the axioms (U1)$\backsim$(U8) hold.

Proof. In view of space constraints, we give a sketch proof for only a few of the axioms. The full proofs are presented in a longer version of this paper which is available from our webpage.

- **(U2)**: We need to prove that if $[\![P]\!]$ implies $[\![F]\!]$ then $[\![P + F]\!]$ is equivalent to $[\![P]\!]$.

– The $\rightarrow$ direction. For each variable, x, there are two cases: $x \in V_F$ or $x \in V_P \setminus V_F$. In the latter case, $[\![P]\!]$ is trivial (from the third line of $[\![P+F]\!]$ in Lemma 1). The former case splits into two further cases:

1. F is triggered. In this case, we know that, for every $\alpha \rightarrow B$ clause for x in $[\![F]\!]$ (Definition 5) there is a $\phi \rightarrow A$ clause for x in $[\![P]\!]$ (Definition 4) such that $\phi \rightarrow A$ implies $\alpha \rightarrow B$ (because $[\![P]\!] \rightarrow [\![F]\!]$). As the ϕ-expressions 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 $A \rightarrow B$ holds, and because A and B are complete over the same vocabulary, $A \rightarrow B$ iff $A \leftrightarrow B$. 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.
2. F defers to P. Again, $[\![P]\!]$ is trivial in this case (from the second line of $[\![P+F]\!]$).

– The $\leftarrow$ direction. Again, the main non-trivial case is 1. as before, i.e. for $x \in V_F$ 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 $[\![P+F]\!] \leftrightarrow [\![P]\!]$.

- **(U5)**: For this, we have defined the operation and on two features, F_1 and F_2, and proved Lemma 2.

Definition 7 (The and Operation on Features) *Let F_1 and F_2 be defined as follows:*

$$F_1 = \{x_1 : \{\alpha_{11} \rightarrow B_{11}, \ldots, \alpha_{1m_1} \rightarrow B_{1m_1}\}, \ldots, x_n : \{\alpha_{n1} \rightarrow B_{n1}, \ldots, \alpha_{nm_n} \rightarrow B_{nm_n}\}\}$$
$$F_2 = \{x_1 : \{\gamma_{11} \rightarrow C_{11}, \ldots, \gamma_{1l_1} \rightarrow C_{1l_1}\}, \ldots, x_n : \{\gamma_{n1} \rightarrow C_{n1}, \ldots, \gamma_{nl_n} \rightarrow C_{nl_n}\}\}$$

Suppose $x_i \in V_{F_1} \setminus V_{F_2}$. Then:

$$(x_i : \{\alpha_{ij} \rightarrow B_{ij} \mid 1 \leq j \leq m_i\}) \in (F_1 \mathsf{and} F_2)$$

Suppose $x_i \in V_{F_2} \setminus V_{F_1}$. Then:

$$(x_i : \{\gamma_{ik} \rightarrow C_{ik} \mid 1 \leq k \leq l_i\}) \in (F_1 \mathsf{and} F_2)$$

Suppose $x_i \in V_{F_1} \cap V_{F_2}$. Then:

$$(x_i : \{\alpha_{ij} \wedge \gamma_{ik} \rightarrow E(B_{ij}, C_{ik}) \mid 1 \leq j \leq m_i, 1 \leq k \leq l_i\} \cup \{\alpha_{ij} \wedge \neg \bigvee_{k=1}^{l_i} \gamma_{ik} \rightarrow B_{ij} \mid 1 \leq j \leq m_i\} \cup \{\neg \bigvee_{j=1}^{m_i} \alpha_{ij} \wedge \gamma_{ik} \rightarrow C_{ik} \mid 1 \leq k \leq l_i\}) \in (F_1 \mathsf{and} F_2)$$

where $E(X,Y) = \begin{cases} X & \text{if } X = Y \\ \emptyset & \text{if } X \neq Y \end{cases}$

Lemma 2 $[\![F_1 \mathsf{and} F_2]\!] \leftrightarrow [\![F_1]\!] \wedge [\![F_2]\!]$

- **(U7),(U8)**: For these proofs, we have defined the operation or on two programs, P_1 and P_2, and on two features, F_1 and F_2 and proved Lemma 3.

Lemma 3 $[\![P_1 \mathsf{or} P_2]\!] \leftrightarrow [\![P_1]\!] \vee [\![P_2]\!]$ *and* $[\![F_1 \mathsf{or} F_2]\!] \leftrightarrow [\![F_1]\!] \vee [\![F_2]\!]$.

5 CONCLUSIONS AND FUTURE WORK

We have formulated SMV and its feature construct in propositional logic and proved that the eight rationality postulates for update hold in this context. This formalises a relationship which had been recognised previously, but never investigated in depth. Now we have established that this relationship exists between the two areas, there are various directions in which we could take our work. We would like to further generalise what we have done to include the TREAT clauses of the feature construct. We could also look at other approaches to feature integration.

Another idea is to investigate whether the knowledge that feature integration is an update operation enables us to predict exactly how the system will be altered when a feature is integrated. We intend to draw on the representation theorems for update (in [5]) to provide a sound semantics for feature integration. This may, in turn, help us to predict how features will interact. Furthermore, this will enable us to develop a classification of features in terms of how disruptive they are to the base system.

ACKNOWLEDGEMENTS

We would like to thank Pierre-Yves Schobbens, Jon Rowe, Dimitar Guelev and Nikos Gorogiannis for fruitful discussions and suggestions.

REFERENCES

[1] M. Calder and E. Magill, eds. *Feature Interactions in Telecommunications and Software Systems*, volume VI. IOS Press, 2000.

[2] P. Gärdenfors, 'Belief revision: An introduction', in *Belief Revision*, ed., P. Gärdenfors, volume 29 of *Cambridge Tracts in Theoretical Computer Science*, 1–28, Cambridge University Press, (1992).

[3] S. Gilmore and M. Ryan, eds. *Language Constructs for Describing Features*. Springer-Verlag, 2001.

[4] M. Huth and M. Ryan, *Logic in Computer Science: Modelling and reasoning about systems*, Cambridge University Press, 2000.

[5] H. Katsuno and A.O. Mendelzon, 'On the difference between updating a knowledge base and revising it', in *Belief Revision*, ed., P. Gärdenfors, volume 29 of *Cambridge Tracts in Theoretical Computer Science*, 183–203, Cambridge University Press, (1992).

[6] K. Kimbler and L.G. Bouma, eds. *Feature Interactions in Telecommunications and Software Systems*, volume V. IOS Press, 1998.

[7] D. Makinson and P. Gärdenfors, 'Relations between the logic of theory change and nonmonotonic logic', in *The Logic of Theory Change*, eds., A. Fuhrmann and M. Mourreau, number 465 in Lecture Notes in Artificial Intelligence, Springer Verlag, (1991).

[8] K.L. McMillan, *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.

[9] K.L. McMillan, *The SMV System (in postscript)*, June 1998. Available at http://www-cad.eecs.berkeley.edu/~kenmcmil/.

[10] M.C. Plath and M.D. Ryan, 'Feature integration using a feature construct', *Science of Computer Programming*, **41**(1), 53–84, (2001).

[11] H. Velthuijsen, 'Issues of non-monotonicity in feature-interaction detection', in *Feature Interactions in Telecommunications Systems*, eds., K.E. Cheng and T. Ohta, volume III, pp. 31–42. IOS Press, (1995).