Computing Minimal Changes of Models of Systems

Nikos Gorogiannis

Supervisor: Mark D Ryan

A PhD thesis submitted to
the University of Birmingham

School of Computer Science
The University of Birmingham
June 2003

Abstract

Minimal change of models of systems is a concept connected to well-known areas of the literature such as theory change and non-monotonic reasoning. However, it seems that little work has been done on bringing together the intuitions from these areas and implementing them in concrete case studies.

Already existing proposals from theory change may be used for minimally changing systems in a variety of ways. However, implementations for these proposals are lacking. We address this gap by employing Binary Decision Diagrams, and construct algorithms that compute these minimal changes. The performance of these algorithms is studied theoretically. In addition, they are used in an application, fault diagnosis of combinational circuits. A case study is examined and benchmarks for the BDD algorithms are obtained.

Furthermore, a specific kind of minimal change is proposed and studied, one that extends expressiveness by using modal and temporal logics instead of propositional logic. This method, minimal refinement, can be used for providing refinements of a system that satisfy a given new requirement, and exhibit as many behaviours of the original system as possible. We then proceed to prove that minimal refinement has interesting theoretical characteristics and that the related algorithmic problems are decidable.

Contents

List of Definitions

List of Lemmas and Theorems

List of Figures

Chapter 1

Introduction

1.1 Models of systems and their changes

A model, in general, is something that hinges on the notion of *representation*: to say that something is a model of something else is to say that the former somehow represents the latter. This quality of models, i.e. the ability to represent other things, such as physical or even abstract entities, allows us to examine and experiment with the models *as if* we were examining or experimenting with the actual entities represented.

Perhaps that is why the notion of a model is such a pervasive concept in science. Within science, models are used to represent the object of enquiry while being expressed in a formal language that allows their manipulation and examination. In Computer Science, models appear in diverse forms and are used to represent and reason about a multitude of things such as systems, programming languages and abstract mathematical structures.

The use of models for representing systems in Computer Science has produced many successful cases where, by using structures expressed in appropriate formalisms to represent actual systems, it has been possible to understand and predict the relevant behaviours of the systems modelled. Such examples include:

- Boolean circuits (also known as combinational circuits) are electronic systems that are ubiquitous and have attracted a significant amount of research. Their input-output behaviour can be effectively modelled by formulae in propositional logic: the input signals are represented by atomic variables, logical gates by the corresponding logical connectives, and the output by the truth value of the formula.

  Moreover, the reasoning capabilities a logical formalism (such as propositional logic) offers allow us to prove facts about the system in question. For example, the question of how two such systems will behave when the outputs of one are connected to the inputs of the other may be addressed. The ability to predict the ensuing behaviour leads naturally to modular design methodologies: the designers can focus on small manageable units that will be put together to assemble a more complex but still manageable and predictable system.

- Another formalism which is extensively used to model systems is the state transition system. Whenever the notion of state appears in the abstraction of the system, or whenever the system under consideration has a temporal behaviour we are interested in studying, then the state transition system may be an appropriate formalism. Examples of systems that have been modelled by this formalism include controller systems such as the control-logic module of a vending machine or an elevator, microprocessors and algorithms such as scheduling or cryptographic protocols.

  Again, this formalism lends itself directly to a logical approach, such as modal logic [BdRV01, Che80] or temporal logic [BCM+90, CES86]. In this way, it is possible to check whether a system will satisfy a requirement that has a non-trivial temporal character. Such a process is a form of *verification*, i.e. the act of ascertaining beyond doubt that a system has a particular property. Similarly, *automatic verification* refers to the process of verifying a system but by using an automated, usually computer-assisted, method. Research in the area of automatic verification has provided us with feasible methods for checking automatically whether a state transition system satisfies a requirement (see e.g. [CE81, BCM+90, McM93]). Therefore, systems like the above-mentioned ones can be verified and debugged before actually manufacturing an instance of them.

Systems may change or be changed over time. This may be the effect of a deliberate action, such as the fitting of a new control-logic sub-module in a vending machine, or it may be the result of forces outside our control, such as a fault developing in a logical gate inside an electronic component. A natural question arises, then, of whether we can apply the formalisms used to model the systems in modelling these changes too. This would extend our ability to reason about systems, by complementing the existing abilities with that of reasoning about change.

Change, however, may also occur in the models themselves, without reflecting some physical phenomenon. Such changes occur frequently when models are used as *designs*. For example, during the process of designing a system, and before that design is ever implemented, it may happen that a shortcoming is diagnosed in the current stage. Then, the existing design is changed to incorporate a work-around for the bug. Also, when incrementally creating a model, it may be desirable to produce successive designs that incorporate an increasing subset of the required properties.

In the examples above, such changes may happen:

- When a logical gate in a combinational circuit develops a fault, its input-output behaviour changes so that the output becomes partially or totally independent of the input: for example, the output may become a constant, in which case the faulty gate is said to be 'stuck-at-x' where x is the output value; or, the output may vary with no restriction.

  When using formulae of propositional logic to model circuits, such faults are readily represented by substituting the corresponding subformula with an appropriate logical constant, in the case of stuck-at-x faults, or a new propositional variable, in the case of unrestricted behaviour.

  In this way, we can re-use the reasoning abilities of propositional calculus in order to answer questions of the form "what are the possible outputs given a specific input and the fact that gate y is stuck-at-1?"

Examples of changes in state transition-based modelling include:

- The act of adding a new component to an existing configuration of a system is called *feature integration*. This process refers to the physical alteration of a system such as the addition of a physical component as well as to non-physical alterations such as the addition of a software 'plug-in'. The corresponding field of research studies, among other things, the ensuing behavioural changes. One example of such work is due to Plath and Ryan [PR99, PR01], where feature integration is formalised as a disciplined alteration of the original state transition system. Systems examined under this approach include the logic controller of an elevator and the telephone system.

- When designing a system, high-level abstractions of the system are usually employed in order to ease verification. It may be desirable, at some point, to derive a more concrete model of the system, one that *implements* the existing model. The relation of implementation between the original model and the derived one is often explained in terms of behaviour-containment: that all the behaviours exhibited by the derived model are exhibited (or allowed) in the original one. This relation is called *refinement* [AL91] and usually entails some kind of language inclusion, depending on the logic used.

1.2 Minimal change

Sometimes, in the study of these types of change, it is desirable to define a measure of how significant or detrimental a specific change is. This measure may be expressible as an 'absolute' function that takes two models and returns a measure of how different they are. More generally, it may be expressible as a 'relative' one, in the form of an ordering $\leq$ on models induced by the starting model: given models A, B and C, $A \leq_C B$ means that A is at most as different from C than B is from C.

However this measure of difference may be expressed, it allows for the ordering of models in a such a way so that questions of the form "given a model A and a set of models S, which models of S are the ones that differ the least from A?" become conceptually tractable. The underlying concept in this question is the one of *minimal change*.

The process of minimally changing a model of a system is not an uncommon one. Examples of its use follow.

Fault diagnosis

As mentioned earlier, parts of a system may develop faults through time. These faults eventually may affect the behaviour of the system in an undesirable way. In many cases, it is possible to repair the system by replacing some of its components. Therefore, we need a method which, on the basis of information about the behaviour of the system, would identify the faulty parts. This process is called *fault diagnosis* [Rei87].

Testing the system, i.e. feeding it with many or all the possible inputs and observing its behaviour may not be an option, for example, when these possible inputs are just too many or when the action of testing itself might adversely affect the system. In these cases, the observed behavioural deviation of the system can be the only information available. Hence, the fault diagnosis algorithm used should accept a description of the system and whatever behavioural observations of the system are available, and on the basis of those produce a diagnosis.

In the case of boolean circuits, such a description of the system is the graph of the circuit, which provides information on the type of each gate and the connections between inputs and outputs of the gates. The evidence of abnormal behaviour would be in the form of input and output bit-vectors, where the outputs are not the expected ones.

In this framework, a measure of the impact of faults developing in the system would be related to the set of the faulty gates. This is necessary because there is always at least one trivial diagnosis which is of no interest: that all components are faulty. One possibility is to identify this measure with the *cardinality* of the set of faulty gates; in this case a fault diagnosis would present us with a set of gates which, if taken as faulty, will explain the discrepancy between the observed and the expected behaviour, and which is the *smallest* such set. Alternatively, this measure can be taken to be the set of faulty gates *itself*, and the ordering of this measure as set-inclusion. In this way, the fault diagnosis algorithm should compute a set of sets of faulty gates that explain the observed deviant behaviour and are minimal with respect to set-inclusion.

Minimal change is central in this process in that we begin from the assumption that the system is functioning correctly, and, in the face of an observation implying a faulty system, we change minimally the assumption so that it explains the observed behaviour, thus arriving at a diagnosis.

Feature integration

Features, in contrast to complete systems, are not complete specifications of the behaviour of the system. They specify a *part* of the system's behaviour, and that is the reason why they can be thought of as add-on components. This demonstrates further the fact that integrating a feature with a base system is not a trivial process. It may give rise to unintended effects, a phenomenon known as *feature interaction*. This occurs when the original system and an add-on component (or two add-on components) combine their behaviours in an unpredicted and usually undesired way. Therefore, it is necessary to be able to reason about the results of such an integration.

The link between the notion of minimal change and feature integration can be summarised as follows. In the set of models of all possible systems, the base system is a single point. The feature, on the contrary, as an incomplete behavioural specification, can be represented by a set of models, those which exhibit the behaviour the feature prescribes. Feature integration, of course, does not yield such a set of models. Instead, it returns the model that admits the feature's behaviour and is *closest* to the base system, under some notion of closeness which is appropriate to the application domain. This view of feature integration is further evidence of its non-monotonic nature. Another link to minimal change is presented in an approach to feature integration by Harris and Ryan [HR02], whereby a specific method of feature integration is shown to be a kind of theory change.

Minimal refinement

Suppose that we have constructed an access control system for a database system. To aid in its verification, an abstraction of this system has been designed in the form of a state transition system. Each possible computation path exhibited by the transition system corresponds to an allowable transaction with the database system. At some point, a further rule is imposed so that, for example, a certain kind of transaction is necessarily followed by another. This rule is to be applied to the original system as an add-on and, as such, it is preferable not to re-design the abstraction from scratch, but re-use the initial transition system as a starting point. Since the original system already circumscribes the allowable behaviours, what we are looking for is a refinement of that model that, in addition, satisfies the new rule.

In an alternative scenario, imagine that a model of a system has been constructed and verified with a model checker. It is now desirable to produce a version of this model which is more low-level so that it can be used for the automatic construction of code that implements this system. The new, more ‘concrete’ version of the system may contain implementation-level details and states. Again, we are interested in producing this refined version automatically, while preserving the behaviours of the specification that are admissible by the implementation mechanisms present in the result.

In both these examples, a refinement of a model is sought. However, refining a model yields an implementation, or a more ‘concrete’ version of it, but not necessarily a useful one: depending on the formalism used, it is very frequently the case that trivial models exist that refine almost every other model (specifically, those that exhibit very few, if any, behaviours). More importantly, we are interested in refining the original model but only so much as is necessary in order to satisfy a given guiding property. In this way, behaviours of the original system are only sacrificed if necessary.

Therefore, refinement can be used to order the set of models that refine the original one and satisfy the given requirement. Then, the desired ones are the members of this set that are minimal with respect to the refinement ordering. We call this type of minimal change, exemplified above, *minimal refinement*. This process can be used whenever the designer has a model that circumscribes the allowed behaviours of the system. Then, a new property can be applied, producing a new model that satisfies the new requirement, refines the initial model and exhibits as many of its behaviours as possible.

1.3 Outline of this thesis

The notion of minimal change is not new; the areas of theory change and non-monotonic reasoning have used it extensively. However, these areas are focused on issues such as the evolution of an agent’s beliefs or the non-monotonic nature of the human way of reasoning; they rarely use minimal change in studying the processes related to changing and developing systems. Moreover, there is a lack of research oriented towards computation: the issues of decidability of decision procedures, their complexity and possible implementations are rarely addressed. The main aim of this thesis, then, is to build on the notion of minimal change of models of systems in concrete case-studies while addressing practical issues such as decidability, complexity and implementations.

Chapter 2: review of related literature

In chapter 2, the literature from the field of theory change that is related to the results in this thesis is presented. A brief introduction to Binary Decision Diagrams, a data structure used extensively in chapter 3, is also given. A review of basic definitions and facts about modal and temporal logic (ACTL) is included, in anticipation of chapters 4 and 5.

Chapter 3: implementations of theory change operators from the literature

In this chapter, minimal change is first examined from the perspective of theory change. The focus is placed on *revisions* and *updates*, the two main types of theory change. We develop algorithms based on a data structure known as the Binary Decision Diagram that compute general revisions and updates, when the object language is the propositional calculus.

Subsequently, we examine some well-known proposals in the literature of theory change [Bor85, Sat88, Dal88, Win88]. These proposals consist in specific, alternative ways of performing minimal change, again with propositional logic as the base language. Algorithms that implement these specific proposals are, then, developed. Some insight into the complexity of the algorithms is also gained by producing upper bounds for the time complexity and for the sizes of the data structures used.

Finally, a framework for fault diagnosis of combinational circuits is formulated. The algorithms developed previously are, then, put to use in a case-study of fault diagnosis of an n-bit adder. From this, empirical data on performance are gathered and benchmarks are produced.

Chapters 4 and 5: minimal refinement

While the previously mentioned types of minimal change are interesting in their own right, propositional logic as the specification language leaves a lot to be desired. Although it definitely simplifies questions of decidability, its lack of expressiveness renders it a rather cumbersome and weak language for reasoning about systems. Modal and temporal languages, on the other hand, retain desirable characteristics related to the decidability of several decision problems but also extend expressiveness in a non-trivial way. Therefore, it would be highly rewarding to study minimal change in a more expressive, yet still computationally viable, framework.

However, beyond the extremely general and logic-agnostic formulations of minimal change found in the theory change literature, there is very little research done on such issues with a modal or temporal object language in mind. This approach will not be reproduced here: instead of trying to propose a general meta-theory for minimal change for modal or temporal languages, a specific type of minimal change, useful in the context of automatic verification and design, will be formulated, studied and implemented.

The operation of minimal change studied here is minimal refinement. Its exposition in the above examples is under-specified in that concepts such as behaviour, model and requirement have not been formally defined. For the purposes of introduction, general definitions will be used to present the idea while specific instances of this framework are formally developed fully in chapters 4 and 5.

Let $\mathcal{L}$ and $\mathcal{M}$ be sets, the set of formulae and the set of models, respectively, and $\models \subseteq \mathcal{M} \times \mathcal{L}$ a satisfaction relation. The class of models that satisfies a formula ϕ will be denoted by $\mathrm{mod}(\phi)$. Let $\leftharpoondown \subseteq \mathcal{M} \times \mathcal{M}$ be a preorder[1] on models that represents *refinement*. We will write $M \leftharpoondown N$ when N refines M, and $M \leftrightharpoons N$ when M, N are refinement-equivalent. Then, the following ordering may be defined.

Definition 1.3.1 (The ordering $\leq_M$)
Let M, A, B be models. Then, $A \leq_M B$ iff

1. $M \leftharpoondown A \leftharpoondown B$ or
2. $M \leftharpoondown A$ but $M \not\leftharpoondown B$ or
3. $A \leftrightharpoons B$. □

The aim of this ordering is to provide a measure of how 'concrete' is a refinement of a model: if $A <_M B$ then A refines M and either B refines A (while the converse is not true), or B does not refine M. In the first case A is a less 'concrete' refinement of M than B is. The same applies in the second case but in a trivial sense.

It is not hard to see that this ordering is reflexive and transitive. As such, it is a preorder. Using $\leq_M$, minimal refinement may now be defined in a formal way.

Definition 1.3.2 (Minimal refinement)
Let M be a model in $\mathcal{M}$ and $\phi \in \mathcal{L}$ a formula. We define the minimal refinement operation $*$ as

$$M * \phi = \min_{\leq_M}(\mathrm{mod}(\phi))$$ □

This definition resembles a particular type of theory change known as update [KM92], in that it is a point-wise definition, i.e. the ordering depends on a model rather than an arbitrary theory as is usual in the case of revisions [KM91].

Given such an operation, several questions arise.

- Generally, the existence of minimal elements within a set with respect to an ordering is not guaranteed and, as such, it is not obvious that a minimal refinement will yield any models. Therefore, before studying further a particular instantiation of this framework, the conditions that guarantee the existence of minimal models must be investigated.

[1] I.e., a transitive and reflexive relation.

In order to address this issue, we employ the notion of *stopperedness*, a concept known from the non-monotonic reasoning literature, which is related (but not equivalent) to that of *well-foundedness*.

Definition 1.3.3 (Stopperedness)
An ordering $\leq$ over $\mathcal{M}$ is stoppered over a collection of sets of models $\mathcal{C} \subseteq 2^{\mathcal{M}}$ iff for each $X \in \mathcal{C}$ and any model $A \in X$ there exists a model $B \in X$ such that $B \leq A$ and B is $\leq$-minimal in X. $\square$

- It may be the case that, for a particular choice of a model M and a formula ϕ, there is no model of ϕ that refines M. In that case $\leq_M$ will be flat inside mod(ϕ) in the sense that for any two models $A, B \in \text{mod}(\phi)$, $A \leq_M B$ iff $A \leftrightarrows B$. As a result, $M * \phi = \text{mod}(\phi)$. We call this case trivial:

  Definition 1.3.4 (Non-triviality)
  Let M be a model in $\mathcal{M}$ and $\phi \in \mathcal{L}$ a formula. We call $M * \phi$ non-trivial iff there exists a model N such that $M \leftarrow N$ and $N \models \phi$. $\square$

  Thus, it is of interest to know the conditions that guarantee non-triviality of the results of the operation.

- Suppose that we have designed a model M that we want to minimally refine with a new property ϕ. Since it is not guaranteed that a model of ϕ that refines M exists, before applying minimal refinement we would like to know whether $M * \phi$ is trivial or not. Is there an algorithm that can compute this and under which conditions?

- Having secured the non-triviality of $M * \phi$ and assuming stopperedness holds, the goal, of course, is to *compute* the models that minimally refine M under ϕ. Intrinsic to this problem is the existence of an algorithm that, given two models M and N and a formula ϕ, decides the minimality of N with respect to M and ϕ, i.e. whether $N \in M * \phi$ or not.

- Lastly, in some situations it is desirable to be able to answer conditional queries about the minimal refinement. For example, we may want to know whether a given model M, when minimally refined by ϕ, will entail a certain property ψ. This amounts to the existence of an algorithm for answering queries of the form $M * \phi \models \psi$.

In chapter 4 these questions are addressed with transition systems as models, *simulation* as refinement and modal logic as the language used to phrase requirements. They are also investigated in chapter 5 where state transition systems with fairness constraints serve as models, *fair simulation* as refinement and the logic of requirements is ACTL.

Chapter 6: conclusions and further work

Finally, we conclude and summarise the possible avenues for extending the work presented in this thesis.

1.4 Published work

The results on the implementation of theory change operators with Binary Decision Diagrams appearing in chapter 3 have been published in [GR02a]. The definitions behind minimal refinement and a significant part of the results presented in chapter 4 have appeared in [GR02b].

Chapter 2

Background

2.1 Theory change

2.1.1 Belief revision as theory change

Theory change is perhaps the field of research that has studied most closely the notion of minimal change (as used in this thesis). It is a field that draws from the area of philosophical logic, artificial intelligence and non-monotonic reasoning, among others. Its object, as the name suggests, is the study of how to revise theories in the presence of tokens of information of a different status than the theories themselves.

The term 'theory' accepts many different interpretations. One of those is, for example, scientific theories: their evolution under contradicting knowledge drawn from experiments has been the object of study from philosophers. Another interpretation, and one of the most known and most studied is *belief*. The prototypical example of this approach is an agent holding beliefs about its world, who comes to learn a piece of information that is inconsistent with its previous beliefs. The process undertaken to reconcile the old belief with the new, more accurate, information is called *belief revision*.

This process can be reformulated more precisely as follows. There is (some kind of) an agent that is in an *epistemic state*, which represents its beliefs about the world. The epistemic state need not be true or, in the logics-of-knowledge parlance, represents belief rather than knowledge. A new piece of information is presented to the agent through some other means, and is believed to be true about the world in preference to the agent's current epistemic state. Thus, the epistemic state needs to be revised, or in other words, to be replaced by another one that includes the "new piece of information" and is, in a sense to be defined, as close to the original as possible.

In form, the object of the study of belief revision is an operator $*$ of the type

$$* : \mathcal{E} \times \mathcal{I} \to \mathcal{E}$$

where $\mathcal{E}$ is the set of possible epistemic states and $\mathcal{I}$ is the set of all possible information tokens.

The characteristics of the information tokens are not always made explicit in the work of researchers (see, e.g., [Gär92b] and the example cited within). Thus, the common knowledge that there is no one, definitive way to revise one's beliefs was to be reinforced by the clarification that there are many different types of *belief change*, each of which may be done in a number of ways. The prime example is *belief update* [KM92], whereby the change in the agent's beliefs does not result because of new, more accurate information about a static world (as implicitly assumed in a belief revision). Instead, belief update aims to incorporate information in an epistemic state about a changing world, and thus interrelates with research on theories and logics of action.

Another epistemological issue pertaining to the field is that of how to treat the distinction between "basic" or self-justified beliefs, and "derived" beliefs or beliefs that ultimately have to be justified in terms of the former ones. By way of example, a belief may be held because it is the consequence of another, more firmly held. When the more firmly held is retracted, the question of whether to retract the weaker one or not is posed. To answer it, a group of researchers believe that the *justification* of beliefs should be included, or accounted for, in the representation of the epistemic state. Such a view is called *foundational*. The *coherence* view, on the contrary, holds that if the right belief change operator is found logical coherence of the epistemic state is all that is needed in order to decide such questions. [dV94] argues that the two views are mathematically equivalent.

Lastly, another issue related to the representation of the epistemic state is its qualitative or quantitative nature. For example, it may involve quantitative characteristics such as probability.

2.1.2 The AGM approach

It would not be unfair to say that most of the research in the field of theory change has followed what has been named the AGM approach. This framework, which is considered to be seminal work, was suggested by David Makinson, Peter Gärdenfors and the late Carlos Alchourrón, or "AGM" [AGM85, Gär88].

Their work assumes that epistemic states can be represented by *belief sets*, 'flat' sets of sentences of some logic that are closed under logical consequence. Thus, $\mathcal{E} \subseteq 2^{\mathcal{L}}$, where $\mathcal{L}$ is the language of the underlying logic, and $\mathcal{I} = \mathcal{L}$. In this sense, the AGM approach is trivially a qualitative one. Moreover, there is no distinction between held and derived beliefs, thus subscribing to the coherence view.

They argued that

- There is no 'one' way of revising one's beliefs.
- While there is nothing that can be said *logically* about what revisions should look like, there is an applicable notion of "informational economy" that revisions should comply with.
- Thus, there is a set of postulates, named rationality postulates, that any reasonable revision operator should satisfy.

The AGM theory also considers two more operations and the interrelations between them. *Expansion* (here denoted as $+$) is simply the logical closure of the union of a set of sentences with another sentence, i.e. $K + \phi = \mathrm{Cl}(K \cup \{\phi\})$. The second operation is called *contraction* and is, in a sense, the opposite of revision; instead of trying to incorporate a belief in the epistemic state, the aim is to retract it.

For completeness' sake we list the AGM axioms for revisions below.

K1. For any sentence $\phi \in \mathcal{L}$ and any belief set K, $K * \phi$ is a belief set.

K2. $\phi \in K * \phi$.

K3. $K * \phi \subseteq K + \phi$.

K4. If $\neg\phi \notin K$, then $K + \phi \subseteq K * \phi$.

K5. $K * \phi = \mathcal{L}$ if and only if $\vdash \neg\phi$.

K6. If $\vdash \phi \leftrightarrow \psi$, then $K * \phi = K * \psi$.

K7. $K * \phi \wedge \psi \subseteq (K * \phi) + \psi$.

K8. If $\neg\psi \notin K * \phi$, then $(K * \phi) + \psi \subseteq K * \phi \wedge \psi$.

Much research has been done within this framework. Major topics include:

- In the AGM approach, a belief change operator is implicitly associated with the epistemic state it operates on. In other words, there is no mention of conditions that a belief change operator should satisfy across the set of all epistemic states. The axioms include conditions on the sentential argument, but none on the belief set. Thus, iteration, i.e. revising a revised epistemic state, is a poorly understood issue. For analysis see [FH99, Rot99] and for proposals see [Nay94, KP00].
- Many proposals have been made for specific revision/update operators, most of which use propositional logic as the underlying object logic, usually with a finite set of propositional letters, [Bor85, Win88, Dal88, Sat88, GS88, For89, ZF96, Wil97]. Their computational complexity has also been studied [EG92, Neb96, LS96, CDLS99].

- The interrelations of theory change and other fields are wide-ranging:
  - Belief revision and non-monotonic reasoning [Gär88].
  - Update and counterfactuals [GM95, RS97, Gra98].
  - Theory change and circumscription [LS95].
  - Belief revision and abduction [BB95].
- Work has also been done on providing semantics for the axiomatic characterisation of theory change operators. Examples are:
  - AGM, along with the rationality postulates for revision, suggested a semantics based on *partial meet contraction functions* [AGM85].
  - An alternative, constructive characterisation of revision operators was suggested by Gärdenfors and Makinson in [GM88], that involves orderings of the sentences of $\mathcal{L}$ named *epistemic entrenchment orderings*.
  - Another approach, by Grove [Gro88], concerns a model-based semantics. This approach characterises revision operators by relating them to *systems of spheres*, or certain families of nested sets of models.
  - A very important characterisation of theory change operators appears in [KM89, KM91]. According to this approach, theory change can be seen as a minimisation operation over the set of models of the new information, with respect to a particular ordering of those models.

2.1.3 Semantical approaches on theory change

A long-standing line of research in the fields of non-monotonic reasoning, conditional logics and counterfactuals (see e.g. [KLM90, BMP97]) is one that approaches the object in a semantical way. On the other hand, the AGM approach is very much an axiomatic one. Because of the interrelations between those areas and theory change, it is no surprise that this trend has been extended into theory change too with very concise results.

The key idea behind the semantical approaches in most of these cases is the concept of a preference relation. The following is an excerpt from [vBvEF93].

> In standard logic, valid reference is defined according to Tarksi's well-known schema expressing 'transmission of truth':
>
> "Each model of the premises is also a model of the conclusion".

> There are two 'flat' quantifications in this definition: one over 'all models' for the premises, and inside that, one over truth 'for all members' of the set of premises. A more flexible perspective arises here when models may be distinguishable as to their relevance, desirability or plausibility in a given style of inference.

Grove's approach

One of the first such approaches to theory change is by A. Grove [Gro88], presented briefly here. Let $T \subseteq \mathcal{L}$ be a theory and $S \subseteq 2^{\mathcal{M}}$ a *system of spheres centred on* T, a family of sets of models that satisfies the following conditions:

S1. S is totally ordered by $\subseteq$.

S2. mod(T) is the $\subseteq$-minimum of S.

S3. The class of all models $\mathcal{M}$ is in S.

S4. If ϕ is a sentence in $\mathcal{L}$ and there is any set of models in S intersecting the models of ϕ, then there is a smallest set (with respect to $\subseteq$) in S that intersects mod(ϕ).

Given S1–4, a function $c_S : \mathcal{L} \to 2^{\mathcal{M}}$ can be defined which given a sentence returns the $\subseteq$-minimum set in S that intersects with the models of that sentence. Then, the following operator $*_S$ can be defined

$$T *_S \phi = \mathrm{th}\,(\mathrm{mod}(\phi) \cap c_S(\phi))$$

where th(M) is used to denote the set of sentences that are true on all models that belong to the set M, and accordingly, mod(ϕ) denotes the class of models that satisfy ϕ.

Grove's representation theorem identifies the class of theory change operators that satisfy the AGM revision axioms as the ones that can be constructed by the previous scheme. In his paper Grove suggests that systems of spheres are actually equivalent to certain orderings on models but does not go into further details.

Katsuno and Mendelzon's approach: revisions

Many of the proposals for specific theory change operators that were being published around that time had a definite semantic flavour, e.g. [Bor85, Win88, Sat88, Dal88]. The common denominator in all of these approaches was isolated and formally characterised in the representation theorems of Katsuno and Mendelzon in [KM89, KM91]. We summarise their results here.

Their formulation is concerned with a propositional logic over a finite collection of propositional atoms. In this case, epistemic states can be modelled by formulae rather than arbitrary sets of sentences, thus making theory change a function of the type $\circ : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$. Katsuno and Mendelzon formulate a set of postulates R1–R6 which, for finite languages, are equivalent to the AGM postulates.

R1. $\psi \circ \mu \vdash \mu$.

R2. If $\psi \wedge \mu$ is satisfiable, then $\psi \circ \mu \equiv \psi \wedge \mu$.

R3. If μ is satisfiable, then $\psi \circ \mu$ is satisfiable.

R4. If $\psi_1 \equiv \psi_2$ and $\mu_1 \equiv \mu_2$, then $\psi_1 \circ \mu_1 \equiv \psi_2 \circ \mu_2$.

R5. $(\psi \circ \mu) \wedge \phi$ implies $\psi \circ (\mu \wedge \phi)$.

R6. If $(\psi \circ \mu) \wedge \phi$ is satisfiable, then $\psi \circ (\mu \wedge \phi)$ implies $(\psi \circ \mu) \wedge \phi$.

Let $f : \mathcal{L} \to \mathcal{M} \times \mathcal{M}$ be a function that takes a formula ψ as an argument and returns a total[1] preorder $\leq_\psi$ on the set of models that satisfies the following conditions

F1. $\min_{\leq_\psi}(\mathcal{M}) = \mathrm{mod}(\psi)$ (cf. with S2 above).

F2. If $\psi_1 \equiv \psi_2$, then $\leq_{\psi_1} = \leq_{\psi_2}$.

Note that, since the language is finite, it is guaranteed that for any satisfiable sentence ϕ, $\min_{\leq_\psi}(\mathrm{mod}(\phi)) \neq \emptyset$ (cf. with S4 above). Katsuno and Mendelzon call such functions *faithful assignments*.

Using such a function f, a theory change operator can be defined:

$$\psi *_f \mu = \mathrm{th}\left(\min_{\leq_\psi}(\mathrm{mod}(\mu))\right)$$

In [KM91], Katsuno and Mendelzon proved the following representation theorem that characterises belief revision operators in terms of faithful assignments.

Theorem 2.1.1 (Representation theorem for revisions [KM91]) *A theory change operator $\circ$ satisfies the axioms R1–R6 if and only if there exists a faithful assignment f such that $\circ = *_f$.* □

This formulation is easily seen to be closely related to that by Grove. A system of spheres induces an ordering on interpretations, and vice versa.

[1]Meaning that for any two models M, N, $M \leq_\psi N$ or $N \leq_\psi M$.

Katsuno and Mendelzon's approach: updates

The other type of theory change that Katsuno and Mendelzon have characterised in [KM92], belief update, is susceptible to a similar approach. Again, they consider a finite propositional language, in which a belief update operator is a function $\diamond : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$. Similarly with the case of revision, they list a set of axioms that all belief update operators should satisfy:

U1. $\psi \diamond \mu \vdash \mu$.

U2. If ψ implies μ, then $\psi \diamond \mu \equiv \psi$.

U3. If both ψ and μ are satisfiable, then $\psi \diamond \mu$ is satisfiable.

U4. If $\psi_1 \equiv \psi_2$ and $\mu_1 \equiv \mu_2$, then $\psi_1 \diamond \mu_1 \equiv \psi_2 \diamond \mu_2$.

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 $\psi \diamond \mu_1 \equiv \psi \diamond \mu_2$.

U7. If ψ is complete then $(\psi \diamond \mu_1) \wedge (\psi \diamond \mu_2)$ implies $\psi \diamond (\mu_1 \vee \mu_2)$.

U8. $(\psi_1 \vee \psi_2) \diamond \mu \equiv (\psi_1 \diamond \mu) \vee (\psi_2 \diamond \mu)$.

In this case, a faithful assignment is a function $f : \mathcal{M} \to \mathcal{M} \times \mathcal{M}$ from *models* to *partial* preorders that have the following property

FU. $\min_{\leq_M}(\mathcal{M}) = \{M\}$, for any model M.

The definition of the theory change operator is point-wise:

$$\psi *_f \mu = \text{th}\left(\bigcup_{M \in \text{mod}(\psi)} \min_{\leq_M}(\text{mod}(\mu))\right)$$

Under this framework, the following representation theorem holds:

Theorem 2.1.2 (Representation theorem for updates [KM92]) *An operator $\diamond$ satisfies conditions U1–U8 if and only if there exists a faithful assignment f such that $\diamond = *_f$.* □

The unifying idea behind these formulations is what Katsuno and Mendelzon call a faithful assignment. This concept supplements the forms of apparatus for 'generating' theory change operators, i.e. epistemic entrenchment orderings, partial meet functions, systems of spheres and so on. The appeal of this particular formulation lies in the fact that in some cases it is easier and more concise to define what it means for a model of a logic to be at least as 'close' to a point of reference than another model. This applies especially when the models of the logic can be thought of as *models of systems*, such as Kripke models and, more generally, transition systems.

2.2 Binary decision diagrams

2.2.1 Definitions and basic results

The *Binary Decision Diagram* (BDD) is a graph-based data structure that can represent boolean functions or propositions. BDDs, in the form used in this thesis,[2] were introduced by Bryant [Bry86]. They are widely known because of their use in *model checking*, a hardware and software verification technique which works by exhaustive state-space exploration. In that context, their usage has led to a dramatic improvement in the efficiency of model checking implementations, and therefore in the size of model that can realistically be explored [McM93, BCM$^+$90].

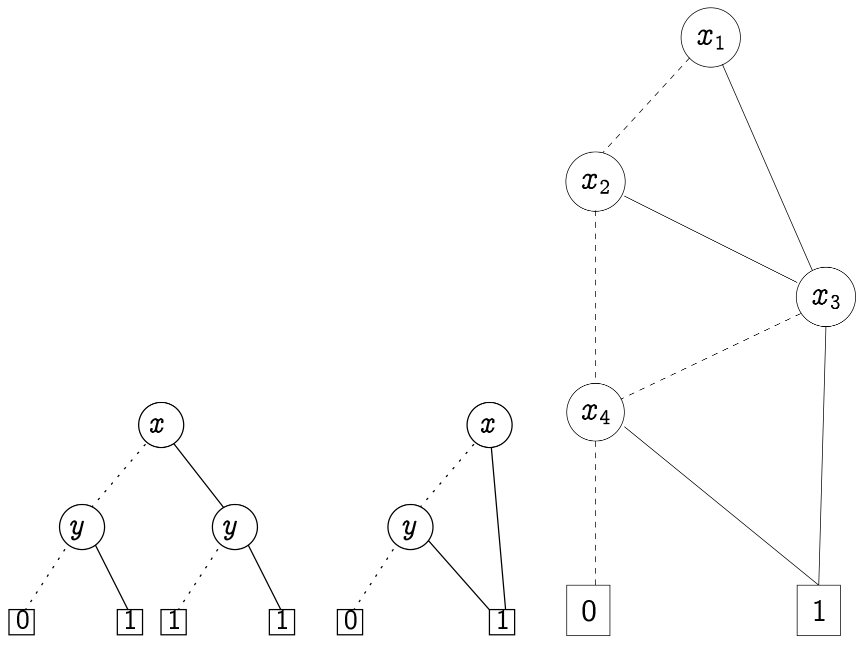

Figure 2.1: (a) The decision tree and (b) the BDD for the formula $x \vee y$. (c) The BDD for the formula $((p \vee q) \wedge r) \vee s$.

The *decision tree* for the formula $x \vee y$ is shown in figure 2.1(a). The dotted lines denote the path to be taken when a propositional atom is false and the solid lines when it is true. The decision tree shows four paths corresponding to the four possible values of x and y, and the leaves show the resulting truth value of the formula in those cases. Decision trees thus code up the truth-table for the formula. They are not space-efficient, having $2^{n+1} - 1$ nodes when the number of atomic propositions in the formula is n.

[2]That is, Reduced Ordered Binary Decision Diagrams or ROBDDs which we will abbreviate as BDDs. Definitions will follow.

The BDD for $x \vee y$ is shown in figure 2.1(b). It is obtained by folding together shared subtrees in the decision tree and removing redundant decision nodes. BDDs can be much more compact than the corresponding decision trees. For example, the BDD for $((p \vee q) \wedge r) \vee s$, shown in figure 2.1(c), contains 6 nodes, while the corresponding tree contains 31 nodes. In the worst case BDDs can still have $O(2^n)$ nodes. However, BDDs have been extensively used in verification where they appear to be a compact representation in practise.

Both decision trees and BDDs assume a fixed ordering of the variables into layers. The size of the decision tree is independent of that ordering, but the size of the BDD is not; the space-economy introduced by sharing sub-diagrams usually depends on the ordering of the variables.

A BDD is fully *reduced* if it has no redundant decision nodes and no isomorphic sub-diagrams. There is an algorithm, called `reduce`, for reducing a decision tree or partly-reduced BDD into its fully-reduced form. `reduce` is efficient, requiring polynomial time in the size of the input structure. Once reduced, BDDs are *canonical*: this means that there is a unique reduced BDD (up to isomorphism) for a given formula with respect to a fixed variable ordering.

Definitions of BDDs and results on their algorithmic complexity can be found in [Bry86]; definitions and a survey of applications can be found in [Bry92]; an extensive analysis geared towards verification in [Som99]; a survey of derivative representations in [Bry95]; and a tutorial on their applications in [And97].

2.2.2 Algorithms on binary decision diagrams

A BDD that represents some propositional formula or boolean function can be manipulated using several algorithms that implement logical operations. Some of these algorithms are presented below along with their complexity characteristics. Note that most of the algorithms presented have an identical space and time worst-case complexity. Thus, unless explicitly stated, complexity will refer to both cases.

Validity, satisfiability and equivalence checking

Because of the canonicity of BDDs, it is easy to check whether a BDD represents a tautology or an unsatisfiable formula. Every tautology is represented by the same BDD, namely, the BDD with a single node, the terminal 1. Thus, validity checking is a constant-time operation. Similarly, a formula is satisfiable if its BDD representation is not the terminal node 0. Again this results in a constant-time operation.

It is well known that the satisfiability problem is NP-complete and the validity problem is coNP-complete. But from the above observations it follows that for deciding these problems it is enough to construct a BDD equivalent to the formula in question. Thus the conversion of a formula to a BDD is NP- and coNP-hard in the length of the formula. Consequently, if it is indeed the case that NP $\neq$ coNP then it follows that the problem of converting a formula to a BDD is not a member of NP $\cup$ coNP.

The canonicity property of BDDs implies that checking if two formulae are equivalent can be done by testing their BDDs for graph isomorphism, which can be done in polynomial time in the sizes of the respective BDDs. In BDD software libraries like CUDD [Som] or BuDDy [LN], hash tables of pointers to BDD nodes are used to enable near-constant time retrieval. In this case, equivalence checking can be done by pointer comparison and hence in near-constant time.

Conversion to binary decision diagrams

Related to the above observation on the complexity of the conversion of a formula to a BDD is the fact that the worst-case space complexity of a BDD is $O(2^n)$ where n is the number of variables in the formula. This occurs when the reduction offers only sub-exponential space-economy. However, the variable ordering drastically affects the amount of reduction possible, so it is not always evident whether this exponential worst-case complexity is due to the ordering or is inherent to the formula represented. On the other hand, many formulae have been found to have very compact BDD representations by choosing appropriate variable orderings [Bry92]. A

Note that the complexity measures used for BDDs depend on the number of variables and not (directly) on the length of the represented formula (which is the usual complexity measure in logic). Because of the canonicity of BDDs, the length of the formula is no more a useful indication of the time and space that BDD algorithms will require. Thus, it makes sense to use the number of variables rather than the length of the formula.

An additional point about the limitations of BDDs is that finding an optimal variable ordering for a given formula, i.e. an ordering that minimises the size of its BDD, is a coNP-complete problem [Bry86]. However, there are several heuristics which perform quite well in practice.

The algorithms `apply`, `negate` and `restrict`

Given two BDDs representing the formulae ϕ and ψ (having $|\phi|$ nodes and $|\psi|$ nodes respectively) together with a binary connective $\bullet$, the algorithm `apply` computes the BDD for $\phi \bullet \psi$. The worst-case complexity of `apply` is $O(|\phi| \cdot |\psi|)$ and it is known to be a tight bound, i.e. there are formulae ϕ and ψ such that their conjunction exhibits a complexity of $O(|\phi| \cdot |\psi|)$ [Bry86].

Given the BDD for ϕ, the algorithm `negate` computes the BDD for $\neg\phi$ by using `apply` and the $\rightarrow$ operator: $\neg\phi = \phi \rightarrow \bot$. Thus its complexity is

$O(|\phi| \cdot 1) = O(|\phi|)$. Note that `negate` could be implemented as a constant-time operation by swapping the terminal nodes. However, for reasons of efficiency, most BDD packages use the same terminal nodes for all stored BDDs, all of which would be negated if the terminal nodes were to be swapped.

These two algorithms provide a way for converting a formula to a BDD without creating the decision tree and then reducing it to BDD form. The BDD representation of a propositional variable is a tree with three nodes, the root labelled by the variable and the two terminal nodes, 1 and 0. Using these and the algorithms `apply` and `negate`, a formula can be recursively converted to the equivalent BDD. Indeed, this is the only algorithm for conversion used in practise since converting a formula to its decision tree is always an exponential operation in the number of variables, whereas conversion using `apply` is expensive only in the worst case.

The result $\phi[C/p]$ of the substitution of a variable p by a boolean constant C can be computed with the algorithm `restrict`. The worst-case complexity is $O(|\phi|)$ [Bry86]. As noted in the same paper, the algorithm can be modified to restrict several variables simultaneously without affecting its complexity.

The algorithms `andExists` and `impliesForall`

The formulae $\forall p.\phi$ and $\exists p.\phi$ are defined as

$$\begin{aligned} \forall p.\phi &= \phi[\top/p] \land \phi[\bot/p] \\ \exists p.\phi &= \phi[\top/p] \lor \phi[\bot/p] \end{aligned}$$

The BDDs for $\forall p.\phi$ and $\exists p.\phi$ can be computed from the BDD for ϕ by the algorithms `apply` and `restrict`, with complexity $O(|\phi|^2)$ [Bry92]. Consecutive quantification over a vector of k variables using this algorithm results in an upper bound for the worst-case complexity of $O(|\phi|^{2^k})$.

McMillan suggests the `andExists` algorithm [McM93] for computing an operation that occurs very often in model checking and which will be used extensively in chapter 3. Let ϕ and ψ be two BDDs over an unprimed and a primed copy of the propositional variables (hence the total number is $2n$, if n is the number of variables). The algorithm computes the consecutive existential quantification over a specified vector of variables, of the conjunction $\phi \land \psi$ but without explicitly forming the BDD for it. An upper bound on the time complexity of this algorithm is $O(|\phi| \cdot |\psi| \cdot 2^{2n})$. However, intuition and empirical evidence both suggest the existence of a smaller bound. The resulting BDD has a size bounded by the general worst-case of the result, i.e. $O(2^{2n-k})$, where k is the number of variables on which we quantify. McMillan also proves that the satisfiability problem can be polynomially reduced to BDD existential quantification over a vector of variables, thus making unlikely the possibility of a polynomial algorithm.

Universal quantification can be computed by making use of the fact that $\forall \equiv \neg\exists\neg$ and the algorithm `negate` giving the same complexity. The dual algorithm to `andExists`, `impliesForall`, is derivable from `andExists` and `negate`, having again the same complexity bound.

The algorithm `replace`

When manipulating BDDs, it is often necessary to replace some variables in a BDD by other variables, an operation corresponding to substitution in logic. This is a linear-time operation if the BDD resulting from the substitution obeys the variable ordering chosen. If it does not, then reordering is necessary and in general this can take exponential time. In this case `andExists` can be used to perform this operation. It is easy to see that when the variables $x'_1, \ldots, x'_n$ do not appear already in ϕ then

$$\phi[x'_1/x_1, \ldots, x'_n/x_n] \equiv \exists x_1, \ldots, x_n . (\phi \wedge (x_1 \leftrightarrow x'_1) \wedge \ldots \wedge (x_n \leftrightarrow x'_n))$$

As such, we can use `andExists` to perform this substitution.

A note on the complexity of BDDs

Problems of high intrinsic complexity exist in the heart of the field of boolean optimisation and data structures for boolean functions. Apart from the high complexity associated with logical problems such as satisfiability and equivalence, there is a plethora of unexpected results about the representational complexity of the various data structures.

Firstly, BDDs can be exponentially more succinct than decision trees. However, there are classes of formulae parameterised over the number of their boolean variables (such as the formula for the middle bit of a multiplier [Bry91]) that have an exponentially-sized best-case complexity. In other words, whatever the choice of variable ordering, the sizes of the BDDs for these formulae increase exponentially in the number of variables. Moreover, by a counting argument it follows that 'most' boolean functions have non-polynomially sized minimal BDD representations: the ratio of the number of boolean functions that can be represented by BDDs of size bounded by a polynomial, to the total number of boolean functions in n variables tends to zero as n tends to infinity [AH97]. Also, studies indicate that a randomly selected boolean function will have a minimal BDD that differs from the worst case by at most a factor of $1 + o(1)$ [GPS98]. Finally, it has been demonstrated that BDDs and resolution perform in fundamentally different ways when used to prove consistency [GZ01]: in this paper classes of formulae are identified where consistency checking by resolution takes exponential time when the BDD method needs only polynomial, as well as classes of formulae where the converse happens.

Secondly, even though it might seem that BDDs are not adequate structures for representing boolean functions, similar results affect even more concise forms of representation. For example, *Boolean Expression Diagrams* (BEDs) [AH97] are another graph-based data structure for representing formulae. They are closely related to boolean circuits. As such, they can be exponentially more succinct than BDDs; there is a boolean circuit for a multiplier that is only polynomially sized in the number of variables. However, BEDs too suffer from exponential 'over-population', as the same counting argument applies to boolean circuits too. Surprisingly, it seems very hard to come up with an example of a function that requires an exponential BED [AH97].

Consequently, it is not reasonable to expect low worst-case (or even average-case) complexities from any such data structure. This belief is reinforced by results on the complexity of problems when graph-based data structures are used to compress the inputs. For example, it has been shown that when BDDs are used to represent graphs, then many graph-related problems suffer an exponential jump in worst-case complexity (e.g. from NP to NEXPTIME) [FKVV99]. This is a phenomenon that affects many other less and more succinct data structures (e.g. boolean circuits).

What complicates matters even more is that BDDs have been used extensively in symbolic model checking, yielding unprecedented performance in many cases. These facts may indicate that the study of worst- and even average-case complexities is not indicative of the usefulness of these data structures. Notably, in the field of symbolic model checking researchers have begun evaluating algorithms with respect to how many *symbolic steps* or pre- and post-image[3] computations are executed, even if each of these steps has exponential worst-case complexity [Som99].

2.2.3 Expression syntax for BDDs

We will use a (slightly bold-face) logical notation to denote the algorithms of the preceding section, as summarised in the table below (B will denote a BDD representing some formula, $|B|$ will be its size in nodes and $\boldsymbol{p}$ is a vector of propositional variables).

algorithm (with arguments)	notation
$\texttt{apply}(B_1, B_2, \circ)$	$B_1 \circ B_2$
$\texttt{negate}(B)$	$\neg B$
$\texttt{exists}(\boldsymbol{p}, B)$	$\exists \boldsymbol{p}.\, B$
$\texttt{andExists}(\boldsymbol{p}, B_1, B_2)$	$\exists \boldsymbol{p}.\, (B_1 \wedge B_2)$
$\texttt{replace}(\boldsymbol{p}, \boldsymbol{p}', B)$	$B[\boldsymbol{p}'/\boldsymbol{p}]$

Some derived algorithms will be presented below, which will be useful in chapter 3. These can be thought of as macros.

[3]Such an operation is, basically, a call to the `andExists` or `impliesForall` algorithm.

BDDs can be thought of representing formulae by representing the set of models[4] that satisfy them. Indeed, all the paths in a BDD that start at the root and end at the terminal 1 are the valuations that satisfy the corresponding formula.[5] To implement some theory change operators, it is necessary to be able to represent relations on models as BDDs. A relation can be thought of as a function which, given two models, returns a boolean value. Therefore, it can be represented as a formula (and thus as the BDD of that formula) over two copies of the atomic propositions, which we call *unprimed* and *primed*, and write as $\boldsymbol{p}, \boldsymbol{p}'$.

For example, consider the ordering $\leq$ shown in figure 2.2 over the four models $\{\overline{p}\,\overline{q}, \overline{p}q, p\overline{q}, pq\}$ of the language $\{p, q\}$. Its BDD is also shown in the figure. To determine whether $M \leq M'$, we supply the truth values $\boldsymbol{p}$ for M and $\boldsymbol{p}'$ for M' to the BDD and get a boolean value result.

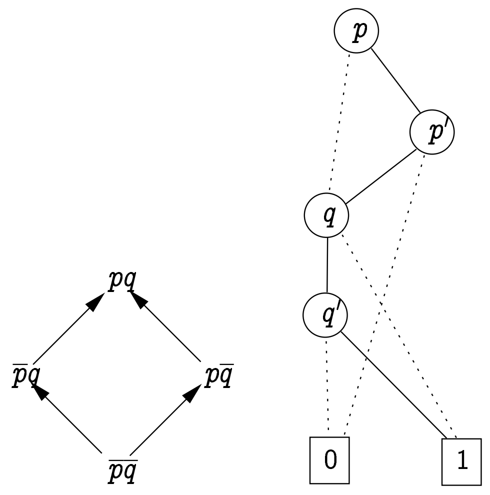

Figure 2.2: An ordering on models of the language $\{p, q\}$, and its BDD.

If B_R is a BDD representing a relation R over unprimed and primed variables, then the BDD for the inverse relation R^{-1} is obtained by simultaneously renaming the unprimed variables to primed and the primed ones to unprimed, i.e. $B_R[\boldsymbol{p}/\boldsymbol{p}', \boldsymbol{p}'/\boldsymbol{p}]$. We write this as $\mathtt{inv}(B)$. The strict counterpart of the relation R, denoted by $R^{<}$, is given mathematically as $R \cap \overline{R^{-1}}$. Thus, the BDD for the strict counterpart is given by

$$\mathtt{strict}(B_R) = B_R \wedge \neg\mathtt{inv}(B_R)$$

[4]Here, and in other places in this thesis, the word 'model' is used with its usual logical meaning, i.e., to mean interpretation. The context of each use of the word should make it clear which meaning is intended (and in some cases both meanings).

[5]Any variables that appear in the formula and not in a path leading to the terminal 1 can be assigned either truth value; all the resulting models will satisfy the formula.

Note that inversion of B_R, i.e. the swapping of the primed and unprimed variables, will necessitate the re-ordering the BDD and is therefore a potentially expensive operation. `inv` is the only instance of variable replacement we will use that does not respect the ordering of variables; all the other replacements can be performed in linear-time.

If R is a relation on a set S and X is a subset of S, we may define the universal post-image of X under R,

$$\text{post}^{\forall}_{R}(X) = \{b \in S \mid \forall a \in X.\, R(a,b)\}$$

The universal pre-image is similarly definable,

$$\text{pre}^{\forall}_{R}(X) = \{a \in S \mid \forall b \in X.\, R(a,b)\}$$

If B_R and B_X are BDDs representing R and X, then the BDDs for these sets are given as

$$\begin{array}{rcl} \texttt{post}(B_R, B_X) & = & (\forall \boldsymbol{p}.\,(B_X \to B_R))[\boldsymbol{p}/\boldsymbol{p}'] \\ \texttt{pre}(B_R, B_X) & = & \forall \boldsymbol{p}'.\,(B_X[\boldsymbol{p}'/\boldsymbol{p}] \to B_R) \end{array}$$

The R-minimal elements of X are defined as

$$\min_R(X) = \left\{b \in X \;\middle|\; \forall a \in X.\, \overline{R^<}(a,b)\right\}$$

This is $\text{post}^{\forall}_{\overline{R^<}}(X) \cap X$, and therefore its BDD can be written in terms of the BDDs B_R, B_X for R and X as

$$\texttt{min}(B_R, B_X) = \texttt{post}(\neg\texttt{strict}(B_R), B_X) \wedge B_X$$

A relation R over a set S is called *total* if for any pair of elements a, b of S, it is the case that $R(a,b)$ or $R(b,a)$. If the relation R is known to be total, then the set of minimal elements coincides with the set of *minimum* elements, i.e. those elements that are less or equal to all other members of the set. Since, in this case, it is not necessary to extract the strict version of R, this permits an optimisation in the way we calculate the BDD for `min`.

$$\min_R(X) = \{a \in X \mid \forall b \in X.\, R(a,b)\}$$

which is the same as $\text{pre}^{\forall}_{R}(X) \cap X$. Therefore the BDD algorithm for `min` need not use `strict`:

$$\texttt{min}(B_R, B_X) = \texttt{pre}(B_R, B_X) \wedge B_X$$

2.2.4 Upper bounds of BDD size and circuit implementations

In chapter 3 bounds for the size of several BDDs will be produced. Instrumental to the derivation of these bounds is a theorem, proved in [McM93], which is presented below.

Let ϕ be an n-ary boolean function and suppose a logical circuit computing ϕ is given. This circuit will contain a number m of blocks that are either gates (binary or otherwise) or primary inputs (inputs are counted as blocks with zero inputs and one output). Let a *linear order* of the circuit be a numbering of the blocks from 1 to m, with the block producing the primary output numbered last. Then, the *forward cross section at block* i is the total number of wires from an output of a block j such that $j < i$ to an input of a block k such that $i \leq k$. The *forward width* w_f *of the circuit* (with respect to the linear order chosen) is defined as the maximum forward cross section for all blocks.

Similarly, the *reverse cross section at block* i is the total number of wires from an output of a block j such that $j > i$ to an input of a block k such that $i \geq k$. The *reverse width* w_r *of the circuit* (again with respect to the linear order) is defined as the maximum reverse cross section at any block. Then, the following theorem holds:

Theorem 2.2.1 (Circuit-based bounds on BDD sizes [McM93]) *If a circuit computing function ϕ has forward width w_f and reverse width w_r for some linear order L, then there is a BDD representing ϕ of size bounded by $n \cdot 2^{w_f 2^{w_r}}$, where n is the number of inputs of the circuit.* □

In what follows, we will only deal with circuits that accept *topological orderings* of their blocks, that is, orderings with $w_r = 0$. In this case, the above-mentioned bound becomes $n \cdot 2^{w_f}$. Note that this bound can yield complexities higher than linear when the forward width is some function of n, rather than just a constant. Another point to note is that changing the boolean base or using aggregate blocks of binary gates as the gates of the circuit mentioned above does not change the order of magnitude of the bound but changes only linearly w_f and w_r.

The numbering of the blocks which is used to calculate w_f and w_r implies an ordering on inputs and that gives us the ordering of the variables in the BDD.

2.3 Modal logic

In chapter 4, we will generally work with a finite set $\mathcal{A}$ of propositional variables. The modal language $\mathcal{L}_{\mathrm{K}}$ of the logic K_m on $\mathcal{A}$ with m modalities is defined inductively;

- if $p \in \mathcal{A}$ then $p \in \mathcal{L}_{\mathrm{K}}$,
- if $\phi, \psi \in \mathcal{L}_{\mathrm{K}}$ then $\neg\phi, \phi \wedge \psi \in \mathcal{L}_{\mathrm{K}}$,
- if $\phi \in \mathcal{L}_{\mathrm{K}}$ then $\Diamond_i \phi \in \mathcal{L}_{\mathrm{K}}$ for all $1 \leq i \leq m$.

The usual propositional abbreviations apply as well as the modal $\Box_i \equiv \neg\Diamond_i\neg$. The usual axiomatisation of K_m follows.

P. Any propositional tautology is an axiom (even those including modalities, e.g., $\Diamond_i p \vee \neg\Diamond_i p$).

K. Any formula of the form $\Box_i(\phi \rightarrow \psi) \rightarrow (\Box_i\phi \rightarrow \Box_i\psi)$ for any $1 \leq i \leq m$.

Its rules of inference are:

MP. Modus ponens: if ϕ and $\phi \rightarrow \psi$, then ψ.

Nec. Necessitation: if ϕ, then $\Box_i\phi$, for all $1 \leq i \leq m$.

The *logic* Λ of K_m is defined to be the smallest subset of $\mathcal{L}_{\mathrm{K}}$ that contains all the instances of the above axioms and is closed under the two rules of inference. The fact that a formula $\phi \in \mathcal{L}_{\mathrm{K}}$ is in Λ is denoted by $\vdash \phi$. If T is a set of sentences and ϕ a sentence, then ϕ is *deducible from* T, written $T \vdash \phi$, if there exists a number $n \geq 0$ and sentences $\psi_1, \ldots \psi_n \in T$ such that $\vdash \psi_1 \wedge \ldots \wedge \psi_n \rightarrow \phi$. T is called *consistent* if $T \nvdash \bot$ and *inconsistent* otherwise.

The most popular semantics for modal logics is through *Kripke models.*

Definition 2.3.1 (Kripke models)
A tuple $M = \langle W_M, r_M, R^1_M, \ldots, R^m_M, v_M \rangle$ is called a Kripke model whenever

- W_M is a set of *states* or *worlds.*
- r_M is a distinguished state in W_M called the *initial state* or the *root.*
- $R^i_M \subseteq W_M \times W_M$ are *accessibility relations.* We will only consider models whose states are all reachable from the root. In other words, for any state $s \in W_M$ there exists a sequence of states $s_1, \ldots, s_n$ such that $s_1 = r_M$, $s_n = s$ and for all $1 \leq j < n$, $(s_j, s_{j+1}) \in \bigcup_{i=1}^m R^i_M$.
- $v_M : W_M \rightarrow 2^{\mathcal{A}}$ is a *valuation* for the propositional letters. $\Box$

By $|M|$ we denote the cardinality of W_M. M is said to be finite if $|M|$ is finite. $\mathcal{K}$ is the class of all Kripke models.

Satisfaction of formulae at a state s is defined inductively by the usual propositional clauses along with the modal one:

$$M, s \models \Diamond_i\phi \quad \text{iff} \quad \exists t \in W_M, (s,t) \in R^i_M \text{ and } M, t \models \phi$$

We will write $s \models \phi$ when the model is obvious. Satisfaction at the level of models is defined as follows.

- $M \models \phi$ iff $r_M \models \phi$,
- $M \models_G \phi$ iff $\forall s \in W_M, s \models \phi$ (*global satisfaction*).

We will write $M \models T$ for a set of sentences T whenever M satisfies all formulae in T. Semantic entailment is defined as follows: if ϕ is a sentence and T a set of sentences, T *semantically entails* ϕ, written $T \models \phi$ iff for all models $M \in \mathcal{K}$, if $M \models T$ then $M \models \phi$. The class of models that satisfies a formula ϕ is denoted by $\mathrm{mod}_{\mathcal{K}}(\phi)$ and the class of models that globally satisfy a formula ϕ, by $\mathrm{mod}^G_{\mathcal{K}}(\phi)$ (similarly for sets of sentences). A set of sentences T is called *satisfiable* iff $\mathrm{mod}_{\mathcal{K}}(T) \neq \emptyset$ (similarly for *globally satisfiable*). The set of sentences true at a state s is denoted by $\mathrm{th}(s)$. The theory of a model is defined as the theory of its root, $\mathrm{th}(M) = \mathrm{th}(r_M)$. Two models M, N are *logically equivalent* iff $\mathrm{th}(M) = \mathrm{th}(N)$.

A few well-known facts about K_m and its logic are summarised below [BdRV01, Che80]. T stands for a set of sentences and ϕ for a sentence of $\mathcal{L}_{\mathrm{K}}$.

- $T \models \phi$ iff $T \vdash \phi$ (*soundness* and *strong completeness* of Λ with respect to $\mathcal{K}$).
- T is satisfiable iff T is consistent.
- If $T \vdash \phi$ and $\phi \vdash \psi$ then $T \vdash \psi$.

We will now turn to the semantical notions of *bisimulation* and *simulation*.

Definition 2.3.2 (Bisimulation)
Let M, N be models and $B \subseteq W_M \times W_N$ a non-empty relation. B is a *bisimulation* if

- It relates the initial states, $(r_M, r_N) \in B$,
- It respects the valuations, $(s,t) \in B$ implies $v_M(s) = v_N(t)$,
- If $(s,t) \in B$ and s' is an R^i_M-successor of s then there exists t', an R^i_N-successor of t, such that $(s',t') \in B$, for all $1 \leq i \leq m$ (the *forth* condition),
- If $(s,t) \in B$ and t' is an R^i_N-successor of t then there exists s', an R^i_M-successor of s, such that $(s',t') \in B$, for all $1 \leq i \leq m$ (the *back* condition). $\square$

If there exists a bisimulation between M, N then M and N are called *bisimilar*, written $M \sim N$ and it follows that $\mathrm{th}(M) = \mathrm{th}(N)$.

Definition 2.3.3 (Simulation)
Let M, N be models and $S \subseteq W_M \times W_N$ a non-empty relation on their state-spaces. S will be called a *simulation* iff it satisfies the first three clauses in the definition of bisimulation, i.e. it must link the initial states, preserve valuations and respect the accessibility relations but in one-way only (the forth condition). $\Box$

If there exists a simulation from M to N we write $M \rightarrow N$ or $N \leftarrow M$ and say that N simulates M or that M is simulated by N. Whenever $M \leftarrow N$ and $M \rightarrow N$ we will say that M and N are *similar* or *simulation equivalent* and write $M \leftrightarrows N$. It is easy to check that simulations are transitive and reflexive.

A formula is called *positive universal* iff it is made up only from propositional letters, their negations, the propositional connectives $\wedge$ and $\vee$ and the universal modality $\Box_i$. $\mathcal{L}_{\mathrm{PU}}$ is the subset of $\mathcal{L}_{\mathrm{K}}$ that consists of positive universal formulae. If s is a state then $\mathrm{PU}(s) = \mathcal{L}_{\mathrm{PU}} \cap \mathrm{th}(s)$. If M is a model, then $\mathrm{PU}(M) = \mathrm{PU}(r_M)$. Dually, a *positive existential* formula is made up from $p, \neg p, \wedge, \vee$ and $\Diamond_i$. $\mathcal{L}_{\mathrm{PE}}$ and PE are defined similarly and are duals of $\mathcal{L}_{\mathrm{PU}}$ and PU respectively. Note that the negation of a PU formula is a PE one and vice versa. If P is a set of PU sentences then P^c is the complement of P with respect to $\mathcal{L}_{\mathrm{PU}}$, i.e. $P^c = \mathcal{L}_{\mathrm{PU}} \setminus P$. $\overline{P}$ is defined as the set that contains the negation of every formula in P, i.e. $\overline{P} = \{\neg\phi \mid \phi \in P\}$.

We define $\Box^*\phi$ to denote a set of sentences as follows,

$$\Box^*\phi = \{\Box_{i_1} \dots \Box_{i_n}\phi \mid \forall n, j, i_j(n \geq 0 \wedge 1 \leq j \leq n \wedge 1 \leq i_j \leq m)\}$$

$\Box^*T$ is defined similarly, but on sets of sentences rather than on formulae:

$$\Box^*T = \bigcup_{\phi \in T} \Box^*\phi$$

If a model satisfies $\Box^*T$ at its starting state then, obviously, it will have to satisfy T on all the states reachable from the root. Therefore, because of the condition that we have imposed on the reachability of all states in a model, it follows that $\mathrm{mod}^{G}_{\mathcal{K}}(T) = \mathrm{mod}_{\mathcal{K}}(\Box^*T)$.

2.4 The logic ACTL

The logic ACTL [GL94] is a fragment of CTL [CES86], a branching-time temporal logic well-known for its use in model checking.

As in the case of modal logic, $\mathcal{A}$ will be a finite set of atomic propositions.

Definition 2.4.1 (The language $\mathcal{L}_{\mathrm{ACTL}}$)
The language $\mathcal{L}_{\mathrm{ACTL}}$ is defined inductively:

- All of the propositional letters $p \in \mathcal{A}$ and their negations are formulae, i.e. $p \in \mathcal{L}_{\mathrm{ACTL}}$ and $\neg p \in \mathcal{L}_{\mathrm{ACTL}}$
- If $\phi, \psi \in \mathcal{L}_{\mathrm{ACTL}}$ then $\phi \wedge \psi \in \mathcal{L}_{\mathrm{ACTL}}$ and $\phi \vee \psi \in \mathcal{L}_{\mathrm{ACTL}}$.
- If $\phi, \psi \in \mathcal{L}_{\mathrm{ACTL}}$ then the formulae
  - $\mathrm{AX}\phi$ (*on all successors, ϕ*),
  - $\mathrm{A}(\phi \mathrm{U} \psi)$ (*on all paths, ϕ until ψ*),
  - $\mathrm{A}(\phi \mathrm{R} \psi)$ (*on all paths, ϕ release ψ*).

  are all formulae in $\mathcal{L}_{\mathrm{ACTL}}$. □

The usual abbreviations apply:

- $\mathrm{AG}\phi \equiv \mathrm{A}(\bot \mathrm{R} \phi)$ (*globally ϕ*) and
- $\mathrm{AF}\phi \equiv \mathrm{A}(\top \mathrm{U} \phi)$ (*eventually ϕ*).

Models for ACTL are structures similar to Kripke models, apart from a few differences.

Definition 2.4.2 (Transition systems with fairness constraints)
A tuple $M = \langle W_M, S_M, \mathcal{A}_M, v_M, \rightarrow_M, \mathcal{F}_M \rangle$ is a model for ACTL with fairness constraints whenever

- W_M is a finite set of states,
- $S_M \subseteq W_M$ is a set of initial states,
- $\mathcal{A}_M \subseteq \mathcal{A}$ is a set of atomic propositions,
- $v_M : W_M \rightarrow 2^{\mathcal{A}_M}$ is a valuation,
- $\rightarrow_M \subseteq W_M \times W_M$ is the accessibility (or transition) relation,
- $\mathcal{F} \subseteq 2^{W_M}$ is a set of fairness conditions. □

It is obvious by the above definition that only finite models will be considered. Also, notice that instead of having just one starting state, now a set of them is provided. This allows for more flexibility, since it allows a non-deterministic choice as to which state the system will begin its execution from. Another point of departure from the modal case is the incorporation of the atomic propositions into the model. Again, this is 'syntactic sugar' that allows more compactness in the descriptions and representations of the models. Notice also that we restrict the number of accessibility relations to one: this relation can be viewed as the union of all the action-representing accessibility relations. As is customary in the temporal logic literature we will ignore these and focus on their union. Moreover, there is no necessity for the states to be reachable by the initial states through the transition relation: since the notion of satisfaction (defined below) is only local to the initial states, there is no danger of 'referring' to unreachable states. Lastly, the fairness conditions are the most important addition; as it will be seen later, from the satisfaction relation of the language, they allow for the exclusion from consideration of some of the computational paths exhibited by the model.

As previously, $|M|$ will denote the cardinality of the state-space of M. The class of all such models will be denoted by $\mathcal{M}_{\mathrm{FTSF}}$, with FTSF standing for finite transition systems with fairness constrains.

Let M be a model. An infinite sequence of states $\pi = s_0 s_1 \ldots$ is a *path* in M iff for all $i \geq 0$, $s_i, s_{i+1} \in W_M$ and $s_i \to_M s_{i+1}$. π^i denotes the i-th state in π. For a path π in M, $\mathrm{inf}(\pi) \subseteq W_M$ is the set of states π visits an infinite number of times. A path π in M is *fair* iff for all $P \in \mathcal{F}_M$, $\mathrm{inf}(\pi) \cap P \neq \emptyset$. Essentially, a path is fair if it visits all the fairness condition sets infinitely often.

Definition 2.4.3 (Satisfaction of ACTL formulae)
Let M be a model and $s \in W_M$ a state. Satisfaction of an ACTL formula on s is defined inductively:

- The usual definitions apply for satisfaction of an atomic proposition, a negated atomic proposition, conjunctions and disjunctions.
- $M, s \models \mathrm{AX}\phi$ if and only if, for all fair paths π in M that start at s, $M, \pi^1 \models \phi$. AX corresponds to the modal $\Box$ with the difference that only the successors along fair paths are considered.
- $M, s \models \mathrm{A}(\phi \mathrm{U} \psi)$ if and only if, for every fair path π in M that starts at s, there is an i such that $M, \pi^i \models \psi$ and for all $j < i$, $M, \pi^j \models \phi$. In other words, ϕ *until* ψ.
- $M, s \models \mathrm{A}(\phi \mathrm{R} \psi)$ if and only if for every fair path π in M that starts at s, for all i, $M, \pi^j \not\models \phi$ for all $j < i$, implies $M, \pi^i \models \psi$. This is the dual of *until* in the sense that in full CTL it is the case that $\mathrm{A}(\phi \mathrm{R} \psi) \leftrightarrow \neg \mathrm{A}(\neg\phi \mathrm{U} \neg\psi)$.

We will write $M \models \phi$ iff for all $s \in S_M$, $M, s \models \phi$ (whether the modal or temporal satisfaction relation is meant by $\models$ will be made clear by the context). Accordingly,

$$\mathrm{mod}_{\mathrm{FTSF}}(\phi) = \{M \in \mathcal{M}_{\mathrm{FTSF}} \mid M \models \phi\}$$

Similarly with modal logic, for any two formulae of ACTL, $\phi \models \psi$ means that for any model $M \in \mathcal{M}_{\mathrm{FTSF}}$, $M \models \phi$ implies $M \models \psi$.

Next, the definition of fair simulation is presented. This concept was introduced in [GL94] where it was called a homomorphism and is also known as $\exists$-simulation [HKR97].

Definition 2.4.4 (Fair simulation)
Let A, B be models with $\mathcal{A}_A \supseteq \mathcal{A}_B$ and α, β be states in A, B respectively. A relation $H \subseteq W_A \times W_B$ is a fair simulation from (A, α) to (B, β) iff

1. $(\alpha, \beta) \in H$
2. For all $a \in W_A, b \in W_B$, $(a, b) \in H$ implies
   - $v_A(a) \cap \mathcal{A}_B = v_B(b)$
   - For every fair path $\pi = a_0a_1 \ldots$ in A with $a_0 = a$ there exists a fair path $\pi' = b_0b_1 \ldots$ in B with $b_0 = b$ such that for every i, $H(a_i, b_i)$.

H is a fair simulation from A to B, denoted $B \leftarrow A$, iff for all $\alpha \in S_A$ there is a $\beta \in S_B$ such that $(\alpha, \beta) \in H$. □

There are several differences with simulation, as seen in the previous section. Firstly, there may be initial states in B that do not correspond to (initial) states in A, something not possible with simulation. Secondly, there may be successors of a state $a \in W_A$ that do not belong to any fair path beginning at a. In that case, these successors do not impose any restrictions in the refinements of A as they necessarily would in the case of simulation.

ACTL consists purely of positive universal formulae. Therefore, fair simulation trivially implies language-inclusion, i.e. $A \leftarrow B$ implies $\mathrm{th}(A) \subseteq \mathrm{th}(B)$. The converse is true as well, since we are only considering finite models.

schemschemss

Chapter 3

Using BDDs to implement Theory Change Operators

3.1 Introduction

An implementation of propositional theory change using a data structure known as the Binary Decision Diagram is presented in this chapter. This work is concerned with the general case of theory change (as defined by sets of axioms) as well as with specific theory change operators from the literature. Upper complexity bounds of these algorithms are produced. In an effort to gain a better understanding of the empirical efficiency of the algorithms involved, a fault diagnosis problem on combinational circuits is presented, implemented and evaluated.

General algorithms are developed for theory change operators that can be defined by faithful assignments for revisions and updates in sections 3.2.1 and 3.2.2. Proposals from the literature of theory change are examined and implemented in sections 3.2.3, 3.2.4, 3.2.5 and 3.2.6. A formulation for fault diagnosis of combinational boolean circuits is described in section 3.3.1. This approach is implemented with BDD algorithms in section 3.3.2, and its experimental performance is analysed in 3.3.3. Finally, work related to this chapter is discussed in section 3.4.

3.2 Theory change operators as BDD algorithms

The approach covered in this chapter is concerned with implementation of theory change operators in a finite, propositional language. As such, belief sets can be represented by single formulae and the framework presented by Katsuno and Mendelzon [KM89, KM91, KM92] applies. Formulae, in turn, will be represented by their corresponding BDDs. The algorithms that will be presented are geared towards the computation of the resulting epistemic state from a theory change operation rather than query answering. In other words, the algorithms will accept two arguments, the epistemic state, ϕ, and the formula to revise with, ψ, and will yield the representation of the resulting epistemic state, $\phi * \psi$, for some theory change operator $*$. In contrast, a query answering algorithm would accept ϕ, ψ, χ as arguments and decide whether $\phi * \psi \models \chi$ without necessarily computing a representation of $\phi * \psi$.

Due to the way BDDs represent the valuations that satisfy the corresponding formula, one may look at them semantically rather than syntactically. In particular, the use of BDDs can serve as a counter-argument to the popular belief in the area of theory change that representing epistemic states by sets of possible models or worlds is less efficient than using belief sets (e.g., [Gär92b]).

Using BDDs to denote the sets of models of formulae enables the direct creation of algorithms implementing the theory change operators that have a straightforward semantical definition. Moreover, since BDDs are canonical, the derived algorithms will be automatically syntax-independent: two logically-equivalent epistemic states should remain logically-equivalent when they are revised by equivalent formulae. This property does not necessarily hold if the theory change operator is defined on a formula-based representation. While some researchers (e.g. [Neb96]) argue that this is desirable, we believe that syntax-dependence should be avoided unless dictated by the nature of the problem to solve.

Of course, the aim is for efficient algorithms implementing theory change. However, the word 'efficiency' has to be used in a relative sense: the problem of propositional theory change is known to be harder than propositional satisfiability [EG92, Neb96, LS96]. Thus, a more precise restatement of this aim is to provide methods that perform well in the 'average' case while inevitably having an intractable worst-case complexity.

3.2.1 Revision defined by faithful assignment

With the aid of the macros described in section 2.2.3, theorem 2.1.1 can be used for computing $\psi \circ \mu$, given a faithful assignment for a revision operator $\circ$. The faithful assignment f is a function which, given a formula ψ, returns an ordering $\leq_\psi$. An algorithm `f` will be assumed, representing f, which takes a BDD over a vector of variables $\boldsymbol{p}$ (for ψ) and returns another BDD over $\boldsymbol{p}, \boldsymbol{p}'$, (for $\leq_\psi$).

By the theorem, $\mathrm{mod}(\psi \circ \mu) = \min_{f(\psi)}(\mathrm{mod}(\mu))$. Therefore, given BDDs B_ψ, B_μ for formulae ψ, μ, the BDD for $\psi \circ \mu$ can be computed as

$$B_\psi \circ B_\mu = \mathtt{min}(\mathtt{f}(B_\psi), B_\mu)$$

where the operator `min` on BDDs for total relations is described in section 2.2.3. Expanding the macros in the above formula gives

$$B_\mu \wedge \forall \boldsymbol{p}'. (B_\mu[\boldsymbol{p}'/\boldsymbol{p}] \rightarrow \mathtt{f}(B_\psi))$$

Suppose that the number of propositional variables is n (in a single copy of the variables), the worst-case time complexity of f is $T_{\mathtt{f}}(\cdot)$ (as a function of the size of the input-BDD) and that the size of the resulting BDD is $|\mathtt{f}(B_\psi)|$. An upper bound for the worst-case complexity of the revision can be computed as follows:

Operation	Time Complexity	Result Size
$B_\mu[\boldsymbol{p}'/\boldsymbol{p}]$	$O(\|B_\mu\|)$	$O(\|B_\mu\|)$
f	$T_{\mathtt{f}}(\|B_\psi\|)$	$\|\mathtt{f}\ (B_\psi)\|$
$\forall \boldsymbol{p}'.(\cdot \rightarrow \cdot)$	$O(\|B_\mu\| \cdot \|\mathtt{f}(B_\psi)\| \cdot 2^{2n})$	$O(2^n)$
$B_\mu \wedge \cdot$	$O(\|B_\mu\| \cdot 2^n)$	$O(2^n)$

Thus, an upper bound of the complexity of the whole operation can be obtained as follows; the overall time complexity is $O(|B_\mu|) + T_{\mathtt{f}}(|B_\psi|) + O(|B_\mu| \cdot |\mathtt{f}(B_\psi)| \cdot 2^{2n}) + O(|B_\mu| \cdot 2^n)$. But any complexity function in $O(|B_\mu|)$ will necessarily be a member of $O(|B_\mu| \cdot 2^n)$ and similarly, any complexity function in $O(|B_\mu| \cdot 2^n)$ will also belong to $O(|B_\mu| \cdot |\mathtt{f}(B_\psi)| \cdot 2^{2n})$. Since the complexity of f is unknown, the bound reduces to

$$O\left(T_{\mathtt{f}}(|B_\psi|) + |B_\mu| \cdot |\mathtt{f}(B_\psi)| \cdot 2^{2n}\right)$$

This upper bound measure may not be indicative of the true situation because:

- Empirical evidence in the context of model checking indicates that the practical efficiency of these operations is much better than their worst case.
- This result is based on non-tight, pessimistic upper bounds for the worst-case complexity of the operators involved. It is telling, for example, that for the result size of the `impliesForall` operation, we are forced to consider the worst case possible, with no reference to the input sizes.

3.2.2 Update defined by faithful assignment

Similarly to theorem 2.1.1, theorem 2.1.2 allows for the computation of $\phi \diamond \psi$, given a faithful assignment for an update operator $\diamond$. In the context of theory updates a faithful assignment is a function from models to orderings on models and as such it can be represented by a BDD and not necessarily by an algorithm on BDDs. In particular, the faithful assignment f is represented as a BDD B_f over $\boldsymbol{p}, \boldsymbol{p}', \boldsymbol{p}''$. Given values for $\boldsymbol{p}''$, it becomes a BDD over $\boldsymbol{p}, \boldsymbol{p}'$ representing a partial ordering.

Therefore,

$$\mathrm{mod}(\psi \diamond \mu) = \bigcup_{M \in \mathrm{mod}(\psi)} \min_{\leq_M}(\mathrm{mod}(\mu))$$

may be computed. B_f can be fed its inputs in any order and it is convenient to manipulate its $\boldsymbol{p}, \boldsymbol{p}'$ parameters first. Using the definition of `min` in section 2.2.3 for partial orderings, the following BDD may be calculated first

$$\mathtt{min}(B_f, B_\mu)$$

which, given $\boldsymbol{p}''$ representing M, computes $\min_{\leq_M}(\mathrm{mod}(\mu))$. This BDD is still parameterised by $\boldsymbol{p}''$; what remains is to take the union over all $M \in \mathrm{mod}(\psi)$. The final answer for the BDD representing $\psi \diamond \mu$ in terms of the BDDs B_f, B_ψ and B_μ is therefore

$$\exists \boldsymbol{p}''.\, (B_\psi[\boldsymbol{p}''/\boldsymbol{p}] \wedge \mathtt{min}(B_{\leq_f}, B_\mu))$$

which when expanded gives (where $B_{\not<} = \neg(B_{\leq_f} \wedge \neg B_{\leq_f}[\boldsymbol{p}'/\boldsymbol{p}, \boldsymbol{p}/\boldsymbol{p}']))$

$$\exists \boldsymbol{p}''.\, (B_\psi[\boldsymbol{p}''/\boldsymbol{p}] \wedge \forall \boldsymbol{p}'.(B_\mu[\boldsymbol{p}'/\boldsymbol{p}] \rightarrow B_{\not<}))$$

The simultaneous double replacement in $B_{\not<}$ can be performed as follows. Assume the variable ordering in the BDDs is such that p_i precedes p'_i which in turn precedes p''_i. It will be extended to include a fourth, temporary copy of the propositional variables $\boldsymbol{t}$ which will be placed in the ordering after the variables we want to swap, but before $\boldsymbol{p}''$. It is easy to see, then, that the substitution $B_f[\boldsymbol{t}/\boldsymbol{p}']$ does not necessitate a re-ordering (i.e. it is a simple renaming) and thus can be performed in time $O(|B_f|)$. Similarly, the substitution $B_f[\boldsymbol{t}/\boldsymbol{p}'][\boldsymbol{p}'/\boldsymbol{p}]$ is of the same linear complexity. Lastly, we want to perform $B_f[\boldsymbol{t}/\boldsymbol{p}'][\boldsymbol{p}'/\boldsymbol{p}][\boldsymbol{p}/\boldsymbol{t}]$ to complete the reversal. This last operation requires a re-ordering, so `andExists` may be used to perform it. Let $B = B_f[\boldsymbol{t}/\boldsymbol{p}'][\boldsymbol{p}'/\boldsymbol{p}]$. Then, $\exists \boldsymbol{t}.\, (B \wedge B_\leftrightarrow)$ needs to be computed, where $B_\leftrightarrow$ is the BDD of the conjunction of the bi-implications between corresponding variables in $\boldsymbol{t}$ and $\boldsymbol{p}$. $B_\leftrightarrow$ is easily seen to be of size $O(n)$. An upper bound of the complexity of this replacement is $O\left(|B_f| \cdot n \cdot 2^{3n}\right)$.

Now, we can turn to the worst-case complexity of the whole operation:

Operation	Time Complexity	Result Size
$B_f[\boldsymbol{p}'/\boldsymbol{p}, \boldsymbol{p}/\boldsymbol{p}']$	$O(\lvert B_f\rvert \cdot n \cdot 2^{3n})$	$O(2^{3n})$
$B_{\not<}$	$O(\lvert B_f\rvert \cdot 2^{3n})$	$O(2^{3n})$
$\forall \boldsymbol{p}'.\, (\cdot \rightarrow \cdot)$	$O(\lvert B_\mu\rvert \cdot 2^{3n} \cdot 2^{3n})$	$O(2^{2n})$
$\exists \boldsymbol{p}''.\, (\cdot \wedge \cdot)$	$O(\lvert B_\psi\rvert \cdot 2^{2n} \cdot 2^{2n})$	$O(2^n)$

Thus, an upper bound for the worst-case complexity of the BDD algorithm for update is $O(|B_\mu| \cdot 2^{6n})$.

3.2.3 Borgida's operator

An interpretation N for a propositional language can be thought as a set containing only the propositional atoms that hold in N. The symmetric set-difference $N \triangle M$ of two interpretations N and M is the set containing all the propositional atoms whose values differ in N and in M.

$$N \triangle M = (N \setminus M) \cup (M \setminus N)$$

Given a formula μ and an interpretation N, the set of differences of N and μ can be defined as:

$$\mathrm{diff}(N, \mu) = \{N \triangle M \mid M \in \mathrm{mod}(\mu)\}$$

Borgida introduced a revision operator $\circ_B$ in [Bor85] that orders interpretations according to the set-inclusion of symmetric set-differences. The definition of $\psi \circ_B \mu$ has two parts:

- If $\psi \wedge \mu$ is consistent, then $\psi \circ_B \mu = \psi \wedge \mu$ (c.f. with axiom R2).
- Otherwise, $M \in \mathrm{mod}(\mu)$ is a model of $\psi \circ_B \mu$ if there is a model N of ψ, such that
$$N \triangle M \in \min_{\subseteq}(\mathrm{diff}(N, \mu))$$

Borgida's revision is known to satisfy R1–R5 but not R6 [KM91]. As such, it is not definable by a faithful assignment for revisions, as the representation theorem 2.1.1 dictates. Let us look how this is implemented in BDDs.

Firstly, $B_\psi \wedge B_\mu$ must be computed and checked for consistency. If it is consistent, then this is also the result of the revision.

If $\psi \wedge \mu$ is inconsistent then the models of the revision are

$$\begin{aligned} &\mathrm{mod}(\psi \circ_B \mu) = \\ &= \left\{ M \in \mathrm{mod}(\mu) \;\middle|\; \exists N.\, \left(N \in \mathrm{mod}(\psi) \wedge N \triangle M \in \min_{\subseteq}(\mathrm{diff}(N, \mu)) \right) \right\} \\ &= \{ M \in \mathrm{mod}(\mu) \mid \exists N.\, (N \in \mathrm{mod}(\psi) \wedge \\ &\qquad\qquad \forall L.\, (L \in \mathrm{mod}(\mu) \rightarrow N \triangle L \not\subset N \triangle M))\} \end{aligned}$$

Therefore, the standard BDD algorithms can be used to implement $\circ_B$ if a BDD B_R that represents the formula $N \triangle L \not\subset N \triangle M$ can be constructed. Assuming that this BDD has variables $\boldsymbol{p}, \boldsymbol{p}', \boldsymbol{p}''$ for M, N, L respectively, the BDD algorithm $\circ_B$ implementing Borgida's revision will be:

$$B_\mu \;\wedge\; \exists \boldsymbol{p}'.\, (B_\psi[\boldsymbol{p}'/\boldsymbol{p}] \;\wedge\; \forall \boldsymbol{p}''.\, (B_\mu[\boldsymbol{p}''/\boldsymbol{p}] \;\rightarrow\; B_R))$$

Now, let us turn to the construction of the BDD B_R. The propositional variables will be ordered from 1 to n with an arbitrary ordering. The truth value of the i-th propositional atom of an interpretation M will be denoted as M_i. The symmetric set-difference of two models can be expressed as a boolean operation (where $(M \triangle N)_i$ is the truth value of the i-th propositional variable of the symmetrical set-difference between M and N)

$$(M \triangle N)_i = \neg(M_i \leftrightarrow N_i) = M_i \oplus N_i$$

The symbol $\oplus$ denotes the xor operation. Set-inclusion of the symmetric set-differences can then be expressed as

$$N \triangle L \subseteq N \triangle M \quad \text{iff} \quad \bigwedge_{i=1}^{n} N_i \oplus L_i \to N_i \oplus M_i$$

and consequently, strict inclusion as

$$N \triangle L \subset N \triangle M \text{ iff } \left(\bigwedge_{i=1}^{n} N_i \oplus L_i \to N_i \oplus M_i \right) \wedge \neg \left(\bigwedge_{i=1}^{n} N_i \oplus M_i \to N_i \oplus L_i \right)$$

from which by negation a formula for computing $N \triangle L \not\subseteq N \triangle M$ is derived. This formula can be converted to a BDD B_R. This BDD will represent the ordering induced by $N \triangle L \not\subseteq N \triangle M$, when an N is chosen.

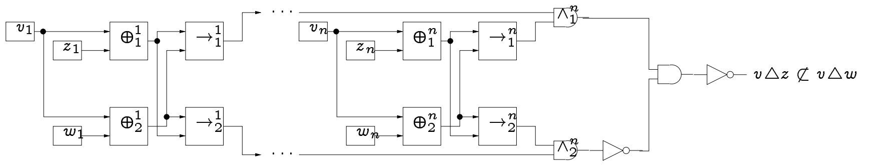

Figure 3.1: Circuit that computes $N \triangle L \not\subseteq N \triangle M$.

Lastly, of interest are the sizes of the BDDs involved and the upper complexity bounds for Borgida's revision as a BDD algorithm. The first step of the algorithm is the consistency check for $B_\psi \wedge B_\mu$, which has a worst-case complexity of $O(|B_\psi| \cdot |B_\mu|)$. If the two formulae are inconsistent then we need to execute the algorithm given above. Central to its complexity will be the size of the BDD B_R. To estimate that, a boolean circuit to compute $N \triangle L \not\subseteq N \triangle M$ is presented in figure 3.1, where $\oplus$ is the xor-gate and $\to$ the implies-gate. The following linear ordering on gates and inputs my be defined (order is from left to right, top to bottom):

$$\begin{array}{lllllllll}
N_1, & L_1, & \oplus_1^1, & M_1, & \oplus_2^1, & \to_1^1, & \to_2^1, & & \\
N_2, & L_2, & \oplus_1^2, & M_2, & \oplus_2^2, & \to_1^2, & \to_2^2, & \wedge_1^2, & \wedge_2^2, \\
\vdots & & & & & & & & \\
N_{n-1}, & L_{n-1}, & \oplus_1^{n-1}, & M_{n-1}, & \oplus_2^{n-1}, & \to_1^{n-1}, & \to_2^{n-1}, & \wedge_1^{n-1}, & \wedge_2^{n-1}, \\
N_n, & L_n, & \oplus_1^n, & M_n, & \oplus_2^n, & \to_1^n, & \to_2^n, & \wedge_1^n, & \wedge_2^n, \\
\neg, & \wedge, & \neg & & & & & &
\end{array}$$

It is easy to check that under this ordering the forward cross section at each gate or input of the circuit is at most a constant: it does not depend in any way on n. Thus by theorem 2.2.1, there exists a BDD B_R representing this circuit of size $O(n)$.

Using this result upper bounds for the complexity of

$$B_\mu \wedge \exists \boldsymbol{p}'. (B_\psi[\boldsymbol{p}'/\boldsymbol{p}] \wedge \forall \boldsymbol{p}''. (B_\mu[\boldsymbol{p}''/\boldsymbol{p}] \rightarrow B_R))$$

may be produced:

Operation	Time Complexity	Result Size
$B_\mu[\boldsymbol{p}''/\boldsymbol{p}]$	$O(\lvert B_\mu\rvert)$	$O(\lvert B_\mu\rvert)$
$\forall \boldsymbol{p}''. (\cdot \rightarrow \cdot)$	$O(\lvert B_\mu\rvert \cdot n \cdot 2^{3n})$	$O(2^{2n})$
$B_\psi[\boldsymbol{p}'/\boldsymbol{p}]$	$O(\lvert B_\psi\rvert)$	$O(\lvert B_\psi\rvert)$
$\exists \boldsymbol{p}'. (\cdot \wedge \cdot)$	$O(\lvert B_\psi\rvert \cdot 2^{2n} \cdot 2^{2n})$	$O(2^n)$
$B_\mu \wedge \cdot$	$O(\lvert B_\mu\rvert \cdot 2^n)$	$O(2^n)$

Therefore, an upper bound of the worst-case time complexity of the BDD algorithm for Borgida's revision is $O(|B_\psi| \cdot 2^{4n})$.

3.2.4 Satoh's operator

Given two formulae ψ and μ, the set of differences of ψ and μ is defined by Satoh [Sat88] as

$$\mathrm{diff}(\psi, \mu) = \bigcup_{N \in \mathrm{mod}(\psi)} \mathrm{diff}(N, \mu)$$

The revision operator $\circ_S$ proposed by Satoh in [Sat88] is defined in first-order logic. Its restriction to finite propositional logic, as described in [KM91], is a 'global' version of Borgida's revision. When revising ψ by μ, instead of considering individually the models of ψ, Satoh's notion of minimality relies on both ψ and μ simultaneously. An interpretation M is a model of $\psi \circ_S \mu$ if there exists a model N of ψ such that $N \triangle M$ is a minimal element of $\mathrm{diff}(\psi, \mu)$. Satoh's revision is known to satisfy R1–R5 but not R6 [KM91].

It is easy to express Satoh's revision as a BDD algorithm, using much of the construction presented above for Borgida's operator. The set of minimal pairs $\min_\subseteq(\mathrm{diff}(\psi, \mu))$ can be expressed as

$$\begin{aligned}\min_\subseteq(\mathrm{diff}(\psi,\mu)) = \{N \triangle M \mid{}& N \in \mathrm{mod}(\psi) \wedge M \in \mathrm{mod}(\mu) \wedge \\ & \forall L \forall K. (L \in \mathrm{mod}(\psi) \wedge K \in \mathrm{mod}(\mu) \rightarrow L \triangle K \not\subset N \triangle M)\}\end{aligned}$$

Therefore the models of the revision are:

$$\begin{aligned}\mathrm{mod}(\psi \circ_S \mu) = \{M \mid{}& M \in \mathrm{mod}(\mu) \wedge \exists N. (N \in \mathrm{mod}(\psi) \wedge \\ & \forall L \forall K. (L \in \mathrm{mod}(\psi) \wedge K \in \mathrm{mod}(\mu) \rightarrow L \triangle K \not\subset N \triangle M))\}\end{aligned}$$

Thus, the BDD algorithm $\circ_S$ will be

$$B_\mu \wedge \exists \boldsymbol{p}'.\,(B_\psi[\boldsymbol{p}'/\boldsymbol{p}] \wedge \forall \boldsymbol{p}'', \boldsymbol{p}'''.\,(B_\psi[\boldsymbol{p}''/\boldsymbol{p}] \wedge B_\mu[\boldsymbol{p}'''/\boldsymbol{p}] \rightarrow B_R))$$

where B_R is a BDD that represents $K \triangle L \not\subset N \triangle M$ (B_R is assumed to contain variables $\boldsymbol{p}, \boldsymbol{p}', \boldsymbol{p}'', \boldsymbol{p}'''$ that correspond to M, N, K, L respectively). This BDD can be constructed from the corresponding formula in the same manner as the one for Borgida's revision.

It is trivial to modify the circuit for Borgida's ordering to produce one that decides $K \triangle L \not\subset N \triangle M$, without changing its forward cross section. Therefore, there is an appropriate variable ordering for which the BDD B_R that represents this ordering is of size $O(n)$. With this result we can compute the upper complexity bounds:

Operation	Time Complexity	Result Size
$B_\psi[\boldsymbol{p}''/\boldsymbol{p}] \wedge B_\mu[\boldsymbol{p}'''/\boldsymbol{p}]$	$O(\lvert B_\psi\rvert \cdot \lvert B_\mu\rvert)$	$O(\lvert B_\psi\rvert \cdot \lvert B_\mu\rvert)$
$\forall \boldsymbol{p}'', \boldsymbol{p}'''.\,(\cdot \rightarrow \cdot)$	$O(\lvert B_\psi\rvert \cdot \lvert B_\mu\rvert \cdot n \cdot 2^{4n})$	$O(2^{2n})$
$\exists \boldsymbol{p}'.\,(\cdot \wedge \cdot)$	$O(\lvert B_\psi\rvert \cdot 2^{2n} \cdot 2^{2n})$	$O(2^n)$
$B_\mu \wedge \cdot$	$O(\lvert B_\mu\rvert \cdot 2^n)$	$O(2^n)$

Thus, an upper bound for the worst-case complexity of $\circ_S$ is $O(|B_\psi| \cdot |B_\mu| \cdot n \cdot 2^{4n})$.

3.2.5 Dalal's operator

The revision operator proposed in [Dal88] takes the distance between two interpretations to be the cardinality of their symmetric set-difference (also known as the *Hamming distance*):

$$d(M, N) = |M \triangle N|$$

where the $|\cdot|$ operator is set-cardinality. The distance of a formula ψ and an interpretation N is defined to be

$$d(\psi, N) = \min\{d(M, N) \mid M \in \operatorname{mod}(\psi)\}$$

Using this notion of distance, a faithful assignment can be defined as

$$M \leq_\psi N \quad \text{iff} \quad d(\psi, M) \leq d(\psi, N)$$

The induced ordering is clearly total, reflexive and transitive, and thus the derived operator is a revision by theorem 2.1.1.

In order to express Dalal's revision as an operation on BDDs, a BDD operation representing the faithful assignment must be constructed. Trying to translate its definition directly into a BDD algorithm would be tedious and most probably highly inefficient, because of the two minimisation operations implicit in $d(\psi, M)$ and $d(\psi, N)$. Instead, the definition is proved below to be equivalent to a somewhat simpler relation between models. Specifically, $d(\psi, M) \leq d(\psi, N)$ if and only if there exists a model K of ψ the distance of which to M is less than or equal to the distance between any model of ψ and N, or in other words,

$$\exists K. (K \in \text{mod}(\psi) \wedge \forall L. (L \in \text{mod}(\psi) \rightarrow |M \triangle K| \leq |N \triangle L|))$$

Left-to-right: if $d(\psi, M) \leq d(\psi, N)$ then $\min\{d(L, M) \mid L \in \text{mod}(\psi)\} = \alpha$ and $\min\{d(L, N) \mid L \in \text{mod}(\psi)\} = \beta$ and $\alpha \leq \beta$. But then there must exist a model $K \in \text{mod}(\psi)$ such that $d(K, M) = \alpha$ and a model $L \in \text{mod}(\psi)$ with $d(L, N) = \beta$. By the definition of β it is obvious that $\beta \leq d(L', N)$ for all $L' \in \text{mod}(\psi)$ and as such, $\alpha \leq d(L', N)$, or $d(K, M) \leq d(L', N)$ for all $L' \in \text{mod}(\psi)$.

Right-to-left: we assume that there exists a model $K \in \text{mod}(\psi)$ such that $d(K, M) \leq d(L, N)$ for all $L \in \text{mod}(\psi)$. Since we are working in a finite propositional language, the existence of minimal models is trivial to prove. Therefore, minimums exist for the sets $\min\{d(L, M) \mid L \in \text{mod}(\psi)\}$ and $\min\{d(L, N) \mid L \in \text{mod}(\psi)\}$, which we call α and β respectively. Obviously, $\alpha \leq d(K, M)$. Moreover, again because of the finiteness of the interpretations, $d(K, M) \leq \beta$ therefore $\alpha \leq \beta$.

Assuming that the ordering $|M \triangle K| \leq |N \triangle L|$ is represented by a BDD B_R with variables $\boldsymbol{p}, \boldsymbol{p}', \boldsymbol{p}'', \boldsymbol{p}'''$ corresponding to M, K, N, L respectively, then the BDD algorithm for the faithful assignment is:

$$\mathtt{f}_D = \exists \boldsymbol{p}'''. (B_\psi[\boldsymbol{p}'''/\boldsymbol{p}] \wedge \forall \boldsymbol{p}''''. (B_\psi[\boldsymbol{p}''''/\boldsymbol{p}] \rightarrow B_R))$$

and the BDD algorithm for Dalal's revision will be

$$B_\psi \circ_D B_\mu = \mathtt{min}(\mathtt{f}_D(B_\psi), B_\mu)$$

where `min` is the macro defined for total relations.

We have not yet produced a BDD for $|M \triangle K| \leq |N \triangle L|$. This will be achieved indirectly while at the same time producing upper bounds for its complexity, by constructing a boolean circuit that computes the result. The translation from the (combinational) circuit notation to boolean formulae is straightforward; then the formula can be converted to a BDD as usual.

This boolean circuit, when given four interpretations M, N, L and K in the form of binary vectors, decides whether $d(M, K) \leq d(N, L)$. Thus, in order to compare $|M \triangle K|$ and $|N \triangle L|$ we need a way to *count* how many propositional variables are true in each set-difference and compare those counts. These counts will be binary numbers representing how many 1s occur in those differences. The maximum number of differences possible is obviously n, thus these binary numbers need only have $k = \lceil \log_2 n \rceil$ bits.

A construction made of n k-bit adders in sequence can be used to do the counting of bits set to 1 in $N \triangle L$ (see left-half of figure 3.2). Blocks labelled

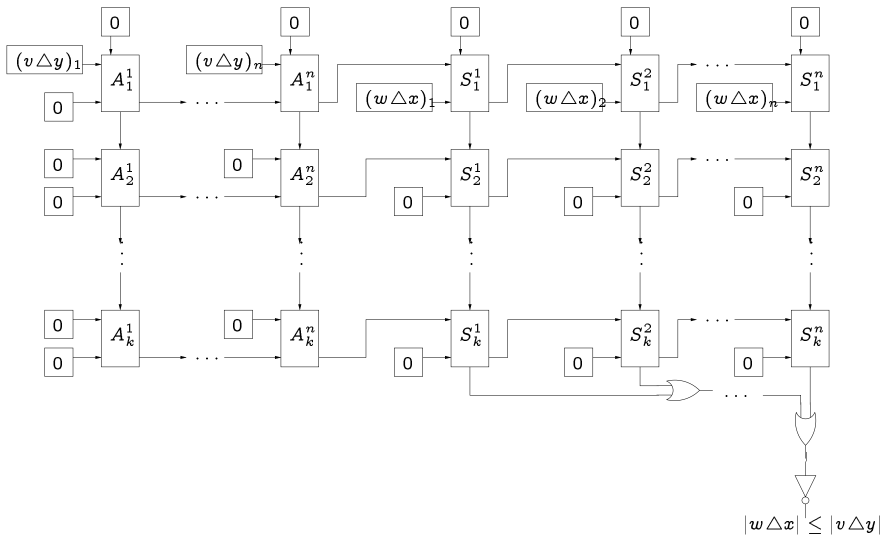

Figure 3.2: Circuit to decide $|M\triangle K| \leq |N\triangle L|$.

A_j^i are *full-adders*. These blocks are simple binary circuits that, given two input bits a, b and a carry bit c, they calculate the sum o and the produced carry bit c':

$$\begin{aligned} o &= a \oplus b \oplus c \\ c' &= (a \wedge b) \vee (c \wedge (a \oplus b)) \end{aligned}$$

Each column in the left-half of figure 3.2 forms a k-bit adder. By connecting zeros to all bits of the first argument except the first one, where $(N\triangle L)_i$ is connected, we ensure that the i-th bit of the difference is added to the second argument, which holds the results of the counting so far.

In order to compare the count we get from the left-half of figure, a structure made from subtracters S_j^i is used in order to *count down* the 1s in $M\triangle K$, seen in the right-half of figure 3.2. Similar to the full-adder, the unit S_j^i is a boolean circuit that given inputs a, b and an input carry c calculates the difference o and the produced carry bit c':

$$\begin{aligned} o &= a \oplus b \oplus c \\ c' &= (b \wedge c) \vee (\neg a \wedge (b \oplus c)) \end{aligned}$$

If, while counting down, the subtraction produces a carry bit then $|M\triangle K| > |N\triangle L|$. Thus, the existence of a carry bit is preserved by taking the disjunction of all carry bits produced by the subtraction stages and by inverting that value the circuit decides $|M\triangle K| \leq |N\triangle L|$.

In order to apply theorem 2.2.1, we define an ordering over the blocks of the circuit:

$$\begin{array}{llllll}
(N\triangle L)_1, & A_1^1, & \ldots, & A_k^1, & & \\
 & & \vdots & & & \\
(N\triangle L)_n, & A_1^n, & \ldots, & A_k^n, & & \\
(M\triangle K)_1, & S_1^1, & \ldots, & S_k^1, & \vee, & \\
 & & \vdots & & & \\
(M\triangle K)_n, & S_1^n, & \ldots, & S_k^n, & \vee, & \neg
\end{array}$$

It is easy to see that on each block, the forward cross section is at most $k+C$ where C is a constant and that the reverse cross section is always zero. Thus, the forward width of the circuit is $k+C$ and the bound given by the theorem is $4n \cdot 2^{k+C} = O(n^2)$, because $k = \lceil \log_2 n \rceil$. Several optimisations can be made on the circuit appearing in figure 3.2, e.g., by replacing blocks with known output with appropriate constants. The forward width of the circuit, however, does not change. Therefore there exists a BDD of size $O(n^2)$ that represents $|M\triangle K| \leq |N\triangle L|$.

We now turn to the worst-case complexity of these BDD algorithms. As noted above, the algorithm for the faithful assignment is

$$\exists \boldsymbol{p}'''.\, (B_\psi[\boldsymbol{p}'''/\boldsymbol{p}] \wedge \forall \boldsymbol{p}''''.\, (B_\psi[\boldsymbol{p}''''/\boldsymbol{p}] \rightarrow B_R))$$

Operation	Time Complexity	Result Size
$\forall \boldsymbol{p}''''.(\cdot \rightarrow \cdot)$	$O(\lvert B_\psi\rvert \cdot n^2 \cdot 2^{4n})$	$O(2^{3n})$
$\exists \boldsymbol{p}'''.(\cdot \wedge \cdot)$	$O(\lvert B_\psi\rvert \cdot 2^{3n} \cdot 2^{3n})$	$O(2^{2n})$

Therefore, an upper bound for the worst-case time complexity of the BDD algorithm for the faithful assignment is $O(|B_\psi| \cdot 2^{6n})$. Thus, in view of the result of section 3.2.1, the derived upper bound for the worst-case time complexity of the revision is $O\left(|B_\psi| \cdot 2^{6n} + |B_\mu| \cdot 2^{2n} \cdot 2^{2n}\right) = O\left(|B_\psi| \cdot 2^{6n}\right)$.

3.2.6 Winslett's operator

Winslett introduced an update operator in [Win88]. The ordering used in this operator is defined using the set-inclusion of symmetric set-differences

$$M \leq_L N \quad \text{iff} \quad L\triangle M \subseteq L\triangle N$$

which is, clearly, a partial order and the mapping is a faithful assignment. As noted in [KM92], Winslett's operator coincides with Borgida's when ψ and μ are inconsistent. In other words, in Winslett's update the second step of the algorithm for Borgida's revision is always used, so the results in section 3.2.3 carry over here unchanged.

3.3 Fault diagnosis

In this section, a formulation of fault diagnosis as a special kind of theory change is presented, along with experimental results gathered from an implementation of the corresponding BDD algorithm. Our goal is not to formulate a fully-fledged theory for fault diagnosis, nor to prove that the best method for fault diagnosis has to use theory change. The aim is to demonstrate the BDD algorithms presented, in a medium-sized example. To that end, we formulate a method for fault diagnosis that works in a well-studied class of systems, combinational boolean circuits, and investigate its complexity in practice.

3.3.1 Fault diagnosis of boolean combinational circuits

Systems can develop faults that make them deviate from their specifications. Given a description of a physical system and an observation of the system (usually, an input-output observation) that is inconsistent with its specification, the problem of fault diagnosis is to deduce which components of the system are faulty.

Since we are interested in discovering which *parts* of the system are malfunctioning, the system description cannot be just of a functional type but must include the description of atomic parts or components. The behaviour of a component will constitute a dependency between its inputs and outputs. However, when such a component is faulty its input-output behaviour is not restricted in any way.

Reiter approached this problem from an abstract point of view in [Rei87]. In his formulation there is an *abnormality predicate* $\mathrm{ab}(x)$ ranging over the set of components. When such an abnormality predicate is false for a component c then c must behave as specified. So, for each component c we have a rule

$$\neg\mathrm{ab}(c) \rightarrow \mathrm{spec}(c, \mathit{obs})$$

where $\mathrm{spec}(c, \mathit{obs})$ is a predicate that is true if and only if the observation obs complies with the predefined behaviour of component c. Essentially, each such rule constitutes the specification of the component it applies to: it is the statement that if the component is functioning normally, then its output will depend in a pre-defined way on its inputs. The set of these rules for all components in a system are called *integrity constraints* of the system.

In the same paper, Reiter points out that his formulation is strongly related to non-monotonic reasoning. As theory change is known to have strong links to non-monotonic reasoning, many researchers have proposed revision as a method of fault diagnosis (e.g., [Win88, Dal88]). The basis of fault diagnosis as theory change is that an observer of the system has two kinds of beliefs about the physical system before observing any of its behaviour:

- that the integrity constraints hold,
- and that no components are faulty.

Now, suppose that a behaviour of the system is observed that is inconsistent with the initial belief. Having chosen an appropriate theory change operator, one may revise the initial belief with the observation, since the observed behaviour implies that the initial belief is false. The intention is that, after revising, the resulting epistemic state should imply which components will explain the observed behaviour if considered to be faulty.

It is obvious that this operation must not retract the part of the initial belief that concerns the integrity constraints, as that would amount to *specification change*, rather than the development of faults in the system. Therefore, the operation must preserve the truth of the integrity constraints and prefer giving up other beliefs while revising. In this case, we shall say that the operation *protects the integrity constraints*.

Since we are interested in implementations of theory change in a finite propositional language, a natural application is fault diagnosis of combinational boolean circuits. These are boolean circuits that do not incorporate components with memory, such as FLIP-FLOPs. Such a circuit consists of a finite number g of unary or binary gates.[1] We define n_I propositional variables I_i corresponding to the primary inputs of the circuit (at most $2g$). For each gate i, we define a propositional variable N_i, its *normality predicate* (which will be the negation of the abnormality predicate mentioned earlier), as well as O_i, its *output value*[2]. The input(s) of each gate will either be primary input(s) or output(s) of other gates. The circuit has also n_{PO} primary outputs, denoted as PO_i, which form a subset of the output values O_i (at most g if no repetitions of results are allowed). Output values of gates not belonging to the set of primary outputs are called intermediate results.

Therefore, for a circuit of g gates we define $n_I + 2g$ (at most $4g$) propositional variables. However, not all valuations of these $n_I + 2g$ variables are possible states the circuit can be found in, even if faulty; if, in some valuation, the normality predicate of a gate is true then its input-output behaviour is fully determined and thus, its output can only assume one value out of the two possible. The set of interpretations allowed under the specification of the circuit is the set of valuations that satisfy its integrity

[1]The presented method can be easily generalised for gates of any (constant) arity and of any (constant) number of outputs.

[2]Since the gate may be faulty, its output value need not be uniquely determined by its inputs. Thus we do need a separate propositional variable for its output value.

constraints

$$\mathrm{IC} = \bigwedge_{i=1}^{g} N_i \rightarrow (F_i \leftrightarrow O_i)$$

where F_i is a boolean expression defining the expected output in terms of the inputs of gate i. The integrity constraints for a circuit that computes $\neg(I_1 \wedge I_2)$, for example, are

$$\mathrm{IC} = (N_{\mathrm{AND}} \rightarrow (I_1 \wedge I_2 \leftrightarrow O_{\mathrm{AND}})) \wedge (N_{\mathrm{NOT}} \rightarrow (\neg O_{\mathrm{AND}} \leftrightarrow O_{\mathrm{NOT}}))$$

The initial belief will be the conjunction of the integrity constraints and of the belief that *all gates are not faulty*:

$$\mathrm{IB} = \mathrm{IC} \wedge \left(\bigwedge_{i=1}^{g} N_i \right)$$

An observation is a description of observed primary input and primary output values

$$\mathrm{OBS} = \bigwedge_{i=1}^{n_I} [\neg] I_i \wedge \bigwedge_{j=1}^{n_{PO}} ([\neg] PO_j)$$

where the notation $[\neg]P$ means that proposition P is optionally prepended with a negation symbol.

The goal is to define a change operator that given an initial belief and an observation of the above forms, returns an epistemic state describing which gates, if taken as faulty, explain the given observation. Of course, the returned formula need not indicate only one combination of faulty gates; there could be several ways in which a faulty circuit can produce a given output.

We define the change operator using a suitable notion of minimality. Interpretations are complete descriptions of the state of the circuit. In other words, an interpretation prescribes the inputs, outputs, intermediate results and normality predicates for the circuit. Intuitively, we want to select all those interpretations that are models of the observed behaviour, while making the *smallest change* to the *persistent* information about the circuit, i.e., the normality predicates. Thus, a suitable notion of minimality is the set-inclusion of differences, but restricted on normality predicates. We do not use a variant of Dalal's operator because that would imply that we are only interested in the minimum *number* of faults necessary to explain the observation.

We choose a variant of Borgida's operator to model this notion of closeness. If M, N, L are interpretations, the ordering is defined as:

$$M \leq_L N \quad \text{iff} \quad \left(\bigwedge_{i=1}^{n_I} I_i(M) \leftrightarrow I_i(N) \right) \wedge \left(\bigwedge_{j=1}^{n_{PO}} PO_j(M) \leftrightarrow PO_j(N) \right) \wedge$$

$$\left(\bigwedge_{k=1}^{g} N_k(L) \oplus N_k(M) \to N_k(L) \oplus N_k(N)\right)$$

where $N_i(M)$ denotes the value of N_i at the interpretation M, and similarly for other propositional variables. Thus, for two interpretations to be comparable, they should imply the same input-output behaviour, hence the first two conjuncts of the above formula. Note that intermediate results do not appear in the definition of the ordering, as they are not observable. The third conjunct formalises our notion of closeness of sets of faulty gates; we are interested in the minimal set of gates (with respect to set-inclusion) that, when faulty, concords with the observation.

With this change operator, protection of integrity constraints is achieved by revising our initial belief not just with the observation, but with the conjunction $\mathrm{IC} \wedge \mathrm{OBS}$.

The result of the revision will include information about the particular observation we revised with, in view of the axiom R2. In particular, the values of primary inputs and outputs in the observation will be included in the resulting epistemic state. Since in fault diagnosis we are only interested in information about the normality predicates of the circuit, we need to eliminate from the resulting belief all knowledge about propositional variables other than normality predicates. We use (boolean) existential quantification to eliminate all propositional variables that carry irrelevant information from the result of the revision. This operation, which in the context of theory change is called *elimination*, is examined in [KM89].

3.3.2 BDD formulation

From the definition of the operation, the models of the revision are

$$\begin{aligned}\mathrm{mod}(\mathrm{IB} \circ (\mathrm{IC} \wedge \mathrm{OBS})) = \{M \mid M \in \mathrm{mod}(\mathrm{IC} \wedge \mathrm{OBS}) \wedge \\ \exists N.\, (N \in \mathrm{mod}(\mathrm{IB}) \wedge \forall L.\, (L \in \mathrm{mod}(\mathrm{IC} \wedge \mathrm{OBS}) \to L \not<_N M))\}\end{aligned}$$

and the models of the diagnosis are obtained by quantifying away all variables except normality predicates.

We will assume that the ordering given previously is represented by a BDD $B_{\leq}$, from which the BDD for the negation of the strict ordering, $B_{\not<}$, can be obtained through the use of `negate` and `strict` ($B_{\leq}$ can be easily constructed as in section 3.2.3). Then, the BDD algorithm for the operation is:

$$\begin{aligned}&\exists \boldsymbol{p}^{\langle I_1,\ldots,I_{n_I},O_1,\ldots,O_g\rangle}.\ ((B_{\mathrm{IC}} \wedge B_{\mathrm{OBS}}) \wedge \\ &\qquad \exists \boldsymbol{p}'.\ (B_{\mathrm{IB}}[\boldsymbol{p}'/\boldsymbol{p}] \wedge \forall \boldsymbol{p}''.\ ((B_{\mathrm{IC}} \wedge B_{\mathrm{OBS}})\,[\boldsymbol{p}''/\boldsymbol{p}] \to B_{\not<})))\end{aligned}$$

Let us now look at the sizes of the BDDs involved as well as at some upper bounds for the complexity of the algorithm.

- Circuits, similar to the one presented in section 3.2.3, that compute the ordering and the negation of its strict counterpart are easily constructible. Thus by theorem 2.2.1, the BDD $B_{\not<}$ representing $\not<$ (where $\leq$ is defined by the formula above) is of size $O(n_I + 2g) = O(g)$.
- The BDD B_{IC} is, of course, dependent on the specific circuit in question. Therefore we cannot give a bound on its size. However, by using the BDD-variable ordering $I_1, \ldots, I_{n_I}, N_1, O_1, \ldots, N_g, O_g$ we ensure an empirically compact representation of B_{IC}.
- The BDD for the conjunction of all normality predicates N_i can be easily shown to have a size of $O(g)$ irrespective of the variable ordering used. Thus the BDD for the initial belief B_{IB} will be of size $O\left(|B_{\text{IC}}| \cdot g\right)$, by the apply algorithm. Similarly, the BDD B_{OBS} for the observation will have a size of $O(g)$ and the conjunction with the integrity constraints $B_{\text{IC}} \wedge B_{\text{OBS}}$ will be of size $O\left(|B_{\text{IC}}| \cdot g\right)$.

Therefore, an upper bound for the worst-case time complexity of the above algorithm is $O\left(|B_{\text{IC}}| \cdot g \cdot 2^{16g}\right)$.

3.3.3 Implementation and experimental results

As mentioned earlier, theory change operations are known to be very expensive in the worst case. Since the complexities reported up to this point concern only the worst-case, and taking into account the fact that the known bounds on BDD operations such as quantification are not tight, we proceeded with a medium-scale implementation of the algorithm presented in the previous section. This implementation can be obtained from `http://www.cs.bham.ac.uk/~nkg/` in source code form. Results regarding the complexity of that algorithm were gathered by trying diagnosis of random observations on a specific combinational circuit, the n-bit adder. Those results along with details about the implementation are presented below.

Implementation

The implementation uses the BuDDy package for the manipulation and construction of BDDs [LN]. As most BDD packages, BuDDy is a C/C++ library that offers services for the creation of BDD variables, the application of operations on BDDs such as those described in section 2.2.3 and statistics reporting on memory usage. BuDDy also offers a C++-only interface that allows for the automation of many tedious book-keeping operations such as BDD-node reference counting, as well as for treating BDDs as first-class objects, rather than just integer indexes that point to a hidden structure inside the BDD library, as is the case with most C interfaces. For these reasons, the C++ interface of BuDDy was used.

The rest of the implementation was coded in C++ and consists of a number of classes designed to embody the functionality for (a) modelling an arbitrary combinatorial circuit and (b) diagnosing that model against an input-output observation that deviates from the circuit's specification. The major modules of the implementation will be described below.

The class `TBddMgr`: This is a singleton class that deals with the initialisation and finalisation of the BDD package, and exports methods for the creation and replacement of named BDD variables (since BuDDy only provides indexed access to BDD variables). It also provides methods for extracting data on the memory usage of BuDDy. Finally, included are methods for the formatting of BDDs in DOT syntax, for visualisation purposes. [3]

Circuit modelling: A class named `Circuit` is used as a base-class for the circuits that will be used for diagnosis; it deals with the basic administration of the integrity constraints of the circuit. The construction phase of an instance of this class uses a number of objects that represent the available types of logical gates: `AndGate`, `OrGate`, `XorGate` and `NotGate` (new types of gates can be easily added by sub-classing the parent class `Gate`). These classes serve as placeholders of the normality predicates and output variables; these are declared automatically as BDD variables during object creation. Adding a `Gate`-type object to a `Circuit` automatically registers the additional integrity constraints and updates the boolean function of the circuit.

Diagnosing circuits: The `Circuit` class also provides the required methods for the diagnosis process. The `buildOrdering` deals with the construction of the BDD that represents the ordering of interpretations described in the previous section. The `diagnose` method accepts an observation in BDD form and performs the diagnosis. It returns a BDD consisting only of normality predicate variables. This BDD may admit several truth assignments for its variables and each one corresponds to a combination of gates that, when faulty, may generate the observation given as an argument to the method.

This set of classes can be used for diagnosis of any combinational boolean circuit. The experiments performed with it are presented in the next part.

Experiments and results

There are (at least) two major factors in this realisation of fault diagnosis:

[3]DOT is a graph description language that the GraphViz tools can automatically convert in a number of picture formats (`http://www.graphviz.org/`).

- We have used belief change to formulate the process of diagnosis. There are other formulations, e.g. ones that are more adapted to particular domains of systems.

- We have used BDD algorithms in order to implement the relevant belief change operation within the presented formulation. Similarly, belief change may be implemented in other ways, without using BDDs to represent formulae.

Understanding the precise effect of the combination of these factors is a major undertaking which falls outside the scope of this research. Instead, we have performed experiments that aim to demonstrate that a typical case of diagnosis, the fault diagnosis of an n-bit adder, can be performed efficiently with this implementation.

The circuit we have chosen for our tests, the n-bit adder, is a circuit that leads to low complexities of the BDDs involved in diagnosis. However, had we selected a circuit that knowingly led to exponential complexities, then the average case for diagnosis (and indeed, worst, best and *any* case) would be provably exponential. Due to time limitations, other circuits with sub-exponential BDD size were selected for further investigation, but not tested (see section 6.1).

The n-bit adder is a combinational circuit that has $2n$ boolean inputs and n boolean outputs.[4] The inputs represent two n-bit integers that are added and output by the adder. The basic building unit in the structure of an n-bit adder is a full adder, already described in section 3.2.5. The purpose of this unit is to add two binary digits along with a carry digit coming from lower-order additions, and produce the resulting sum and carry bits. There are many ways to implement such a circuit; the one used in this section is a standard design shown in figure 3.3.

In order to implement the n-bit adder, a `FullAdder` class was implemented, that automatically builds the full adder shown in figure 3.3, using the classes presented earlier. On top of that, a `NBitAdder` class was built, derived from the `Circuit` class and thus inheriting the methods suitable for the diagnosis of the n-bit adder. The design for the n-bit adder is shown in the right half of figure 3.3.

The BDD for the integrity constraints B_{IC} for the n-bit adder, was found to have linear complexity in n. Therefore, and since each full adder consist of 5 gates, an upper bound for the worst-case complexity of the fault-diagnosis algorithm on the n-bit adder is $O(n^2 \cdot 2^{90n})$, in view of the result in the previous section.

The computer used for the experiments was an UltraSparc-II at 450 MHz with 4Gb of memory, running SunOS 5.8. The large amount of physical

[4]Strictly speaking, an n-bit adder has $n+1$ outputs because of the carry bit indicating overflow, but for the sake of simplicity we will ignore the carry bit.

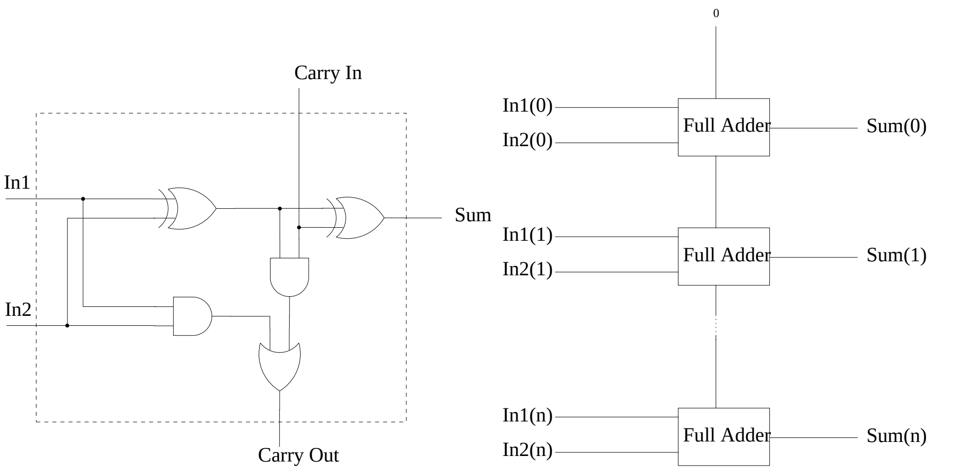

Figure 3.3: A standard design for a full adder (left) and an n-bit adder (right).

memory on the computer ensured that timings of BDD operations would not be deemed inaccurate by virtual memory swapping. The code as well as the BuDDy library were compiled and optimised with the GCC tool-chain[5] for SPARC processors.

The BuDDy package, as most of the BDD packages available, offers heuristics for automatically re-ordering the variables of BDDs in order to attain lower space complexities. This capability is very important in an application for fault diagnosis on arbitrary circuits, since the BDD for the integrity constraints is of an unpredictable size and can benefit from automatic re-ordering. However, having chosen a specific circuit to perform our experiments on, we did not make use of this feature.

We attempted two kinds of tests. In the first one, n-bit adders of successively larger n were generated, and each one was diagnosed with a set of uniformly distributed random observations. The number of possible observations for n bits is 2^{3n}. A constant percentage of those 2^{3n} observations were sampled and fed to the diagnosis algorithm. The space complexity of each diagnosis was recorded and an average complexity for each n was produced.

The reason for sampling the space of all possible observations as inputs to the fault diagnosis algorithm is that, in general, faults affecting even only one gate may affect any number of output bits. Secondly, in the case of the n-bit adder this is not only a possibility but a certainty; if, e.g., one of the xor gates of the very first full adder malfunctions then any output configuration is possible. Also, for generality's sake, no restrictions on the number of faults occurring in the circuit were posed, something reflected in

[5]http://www.gnu.org/gcc/.

our choice of ordering as well; had we used such restrictions, a variant of Dalal's ordering might have been more appropriate.

Unfortunately, sampling a fixed percentage of an exponentially sized population leads to exponential time taken for the tests. Indeed, at 7 bits, diagnosing 10% of the total number of observations possible amounts to running the diagnosis algorithm 209716 times, and for 8 bits this number is multiplied by 8. Due to the excessive time taken for the fixed-percentage tests, we could only run them for up to 7 bits. The results for this test are shown below (space complexity is measured in BDD nodes produced):

Bits	Average Space Complexity	Number of Samples
1	234	1
2	886	7
3	1906	52
4	3229	410
5	4861	3277
6	6804	26215
7	9052	209716

In order to get an idea of the complexities concerned in larger circuits, we tried a second test by running our algorithm on 1000 samples for each bit-size. Admittedly, this approach reduces exponentially the accuracy of our averages in the number of bits. However, by running this test multiple times we have empirically verified that the variance of the results is not significant (in particular, the ratio of standard deviation to average never exceeded 5% in the case of space complexity and 10% in the case of time complexity). The average space complexity (in BDD nodes produced) and the average time complexity (in ms) are shown in figures 3.4 and 3.5 respectively.

We can easily observe that the two curves are very similar. This probably means that most of the time consumed by the algorithm is spent in memory allocation (and possibly deallocation, but in a more or less constant ratio to the allocations). This effect is not unknown for BDD algorithms, whereby many primitive operations perform one allocation per time-step. While the memory-consumption curve is smooth, the time-consumption curve is not. This is most probably due to time sampling issues as it features prominently wherever the times measured are less than 25ms, a value very close to the order of magnitude of the time-slice of the computer system that was used for the experiments.

As with any empirical investigation, these results cannot be taken as conclusive evidence of tractability or intractability. However, we did use a nonlinear least squares method (Marquardt-Levenberg algorithm) to fit a number of classes of functions to the above curves. To our best knowledge, the best fit was a quadratic function, being significantly better that exponential and sub-exponential non-polynomial ones. In addition, in the case

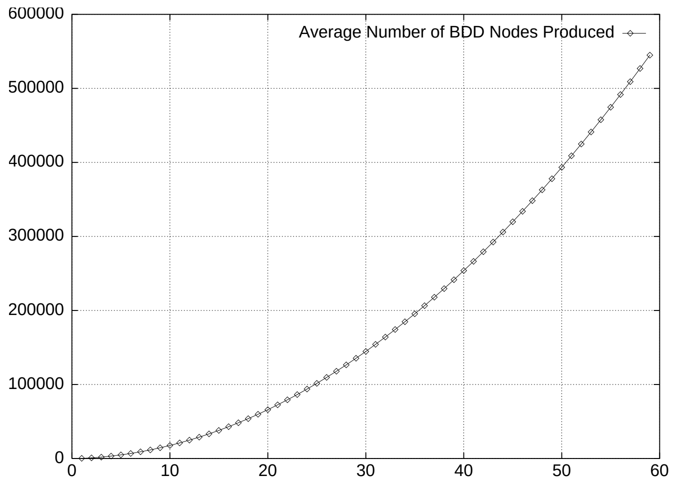

Figure 3.4: Average number of BDD nodes produced per diagnosis in the number of bits.

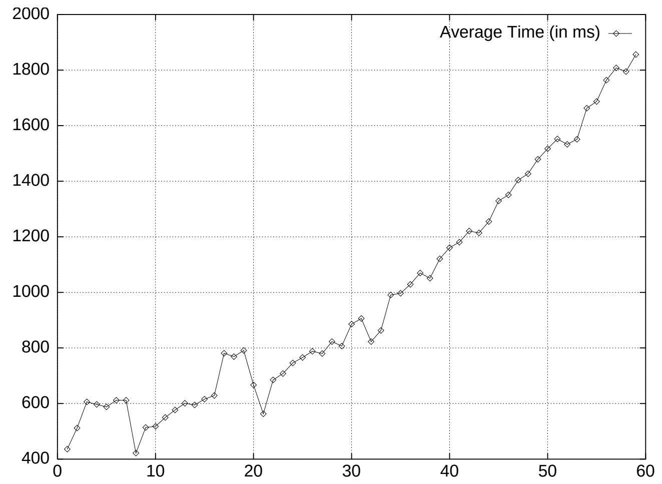

Figure 3.5: Average time (in ms) spent per diagnosis in the number of bits.

of space complexity, the resulting quadratic function from fitting the first or last 15 data points predicts reasonably well the remaining 45. Therefore, it seems plausible that the average complexity of diagnosing a randomly chosen, uniformly distributed, input-output observation on an n-bit adder is of quadratic order of magnitude.

3.4 Related work

Madre and Coudert, in [MC91], use a variation of BDDs called Typed Decision Diagrams (TDGs) to implement what they call a logically complete reasoning maintenance system. TDGs are identical to BDDs apart from the fact that they may incorporate negated edges in their graphs. A negated edge indicates that the pointed subgraph is to have its truth value considered inverted. Their results indicate that Madre and Coudert's TDG algorithms may be used for the following problem. Let KB be a formula and E a conjunction of literals that is inconsistent with KB. The problem is then to compute the set of maximal subsets of the literals of E that are consistent with KB. This could be seen as a variant of belief revision in that the 'initial belief', E, is relaxed in order to achieve consistency with the 'new information' KB. The crucial difference is that KB is not entailed by the result. Madre and Coudert provide experimental evidence that the TDG algorithms they present, performed empirically well.

Würbel, Jeansoulin and Papini in [WJP01] present results on spatial information revision. Their application is related to geographical information systems and the fact that imprecise data measurements make revisions necessary. They encode their data in such a way so that all considered formulae or knowledge bases are always in the form of conjunctions of literals. Three methods of implementing a particular revision operator are considered, one of which uses BDDs. The BDD method makes use of the special form the formulae are in and is essentially a graph search and 'surgery' on the BDD itself. As such, the approach described in [WJP01] is specific to the application considered and therefore not immediately, if at all, generalisable to the other operators found in the literature.

Chapter 4

Minimal refinement and Modal Logic

4.1 Introduction

Imagine that for the purposes of model checking a microprocessor, a transition model has been constructed that models, for example, the interface between the microprocessor (CPU) and the memory management unit (MMU). One way to distinguish between the state transitions of the different subsystems is to have many transition relations in the model, one for each subsystem. Suppose further that an error has been discovered in the model, such that a particular CPU transition is followed by an insufficient number of 'wait-states' before the MMU has reached an appropriate state of its own. Also, the model is considered to be sufficient in other cases. It would be desirable to refine this model into another which does not contain this problematic scenario, possibly by unfolding existing state-loops, while avoiding the addition of 'new' transition sequences. The framework presented in section 1.3 could, then, be used for the automatic generation of such a refinement.

In another scenario, suppose that in order to study mathematically (and possibly model check!) a board game of two players, we wish to construct a transition system that models all or some of the possible evolutions of the game. This transition system would have two transition relations, a black and a white one, each one corresponding to a player. Suppose that we are interested in studying the effect of adding a new rule to the game, of the form "if the white player has captured the centre of the board, then the black player must retreat by one square". Since new game-state sequences should not be added by the incorporation of this rule, what we are really looking for is a refinement of the initial model. Moreover, this refinement should be minimal, as the addition of this rule should only exclude evolutions of the game that are expressly forbidden.

In both of these cases, modal logic can be used to phrase the requirements presented. For example, in the second one, a formula of the form $\mathsf{CentreCapturedBy}(white) \to \Box_{black}\mathsf{Retreats}(black)$ is sufficient. More generally, modal logic can be used as a simple specification language for processes. It provides the mechanisms to describe what the state of a process is, what actions are possible and what actions are necessary from that state. Indeed, modal logic can be seen as Hennessy-Milner logic [HM85] extended with propositional operators and state-information.

In the universe of transition systems and modal logic, a behaviour of a model is a *computation tree*. If refinement is seen as containment of behaviours, then the appropriate concept in this case is *simulation* [Mil71, BFG$^+$91]. This is the notion of refinement with which we will instantiate the ordering $\leq_M$ and minimal refinement in this chapter.

Our main interest is in the applications of minimal refinement; therefore, and especially in the context of model checking, the class of finite models is central to this work due to the fact that finite-state processes can be described in a straightforward way as finite models. However, a detour will be taken through another class of models, before the discussion of the finite case. This class, the modally-saturated models (referred to as m-saturated from now on) is not a class that directly lends itself to practical uses of modal logic, since it includes infinite and uncountable models, and only imposes modest restrictions on its members. On the other hand, since it combines productively with the notion of the ultrafilter construction (more on which will follow), it will provide us with a set of results that will serve as tools for approaching the finite case. Moreover, it possesses the following properties [Hol95]:

- It subsumes the class of image-finite models (and hence the finite ones).
- It has the Hennessy-Milner property. (A class of models has the *Hennessy-Milner property* whenever for every pair of its models, they are bisimilar if and only if they are logically equivalent.)
- It is maximal in the sense that no proper superclass of it has the Hennessy-Milner property.
- It has also been used to provide semantics for simple process algebras.

The outline of this chapter is as follows. Firstly, the definition of minimal refinement is examined in the modal case, and examples motivate the related questions, in section 4.2. Then, minimal refinement is examined in the case of m-saturated models in section 4.3, where the questions of non-triviality and stopperedness are addressed. The case of minimal refinement of finite models is investigated in section 4.4, where the results on stopperedness, non-triviality and decidability of several problems are discussed. Finally, related work is discussed in section 4.5.

4.2 Minimal refinement in the modal case

Definition 1.3.1 is reproduced here. Let M, A and B be models. Then, $A \leq_M B$ if

1. $M \leftharpoonup A \leftharpoonup B$ or
2. $M \leftharpoonup A$ but $M \not\leftharpoonup B$ or
3. $A \leftrightarrows B$.

where $\leftharpoonup$ means simulation.

Notice that the definition of minimal refinement,

$$M * \phi = \min_{\leq_M}(\mathrm{mod}(\phi))$$

is essentially parameterised over two dimensions here. Firstly, there are two meanings for the mod operator, the local and the global, as defined in section 2.3. Secondly, since we are going to be concerned with several classes of models, the mod operation will range over the appropriate class. We will add some notation to make these issues clear. The $*$ symbol and the mod operator will be subscripted with the class of models in question. In addition, the exponent G when it exists will indicate that only the models that satisfy globally the given property are to be considered. Lastly, the notation $[G]$ will indicate that the result under discussion will concern both the local and the global case. So, for example, $\mathrm{mod}^G_{\mathcal{M}}(\phi)$ signifies the subclass of models in the class $\mathcal{M}$ that satisfy ϕ on all of their states.

Therefore, for some class of models $\mathcal{M}$ we have:

$$M *_{\mathcal{M}} T = \min_{\leq_M}(\mathrm{mod}_{\mathcal{M}}(T))$$

and its global version

$$M *^G_{\mathcal{M}} T = \min_{\leq_M}\left(\mathrm{mod}^G_{\mathcal{M}}(T)\right)$$

Accordingly, stopperedness has to be parameterised over local/global satisfaction. Definition 1.3.3 says that an ordering $\leq$ over $\mathcal{M}$ is stoppered over a collection of sets of models $\mathcal{C} \subseteq 2^{\mathcal{M}}$ iff, for each $X \in \mathcal{C}$ and any model $A \in X$, there exists a model $B \in X$ such that $B \leq A$ and B is $\leq$-minimal in X. The collection of sets $\mathcal{C}$ will eventually be defined as the collection of sets of models of some sentence or some set of sentences. As such, these collections will again be ultimately parameterised over global/local satisfaction. Further details will follow in the specific cases.

The questions described in section 1.3 will now be reviewed in the modal case. Firstly, observe the model E in figure 4.1. It essentially represents a system that starts from a p state (the diamond shape indicates the starting state) and which will execute at least one transition, the result of which is a $\neg p$ state, before returning to the p state.

The importance of the question of non-triviality is easily demonstrated: suppose we wanted to compute $E *_{\mathcal{K}} \Diamond p$ (note that this is the local version referring to the satisfaction of formulae at the initial state, over all Kripke models). It should be clear that a model that refines E and satisfies $\Diamond p$ does not exist, and as such, $E *_{\mathcal{K}} \Diamond p = \text{mod}_{\mathcal{K}}(\Diamond p)$. The conditions under which this does not happen, are, then, of interest to us.

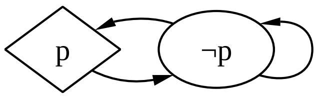

Figure 4.1: E, an example Kripke model.

Suppose, instead, that we would like to compute $E *_{\text{FIN}} \Box\Box\neg p$, where FIN denotes the class of finite Kripke models. In this case, it is easy to show that finite models that refine E and satisfy $\Box\Box\neg p$ indeed exist, as it can be seen in figure 4.2. All the models shown in figure 4.2 except for E, the leftmost, satisfy $\Box\Box\neg p$ on their initial states. Moreover, they form a chain, i.e. a set of models ordered totally by simulation (the dashed lines indicate the simulation relations). It is obvious that there exists an infinite number of finite models that are in between the leftmost model and the second one in terms of simulations, that satisfy $\Box\Box\neg p$ and are progressively smaller with respect to the ordering $\leq_E$.

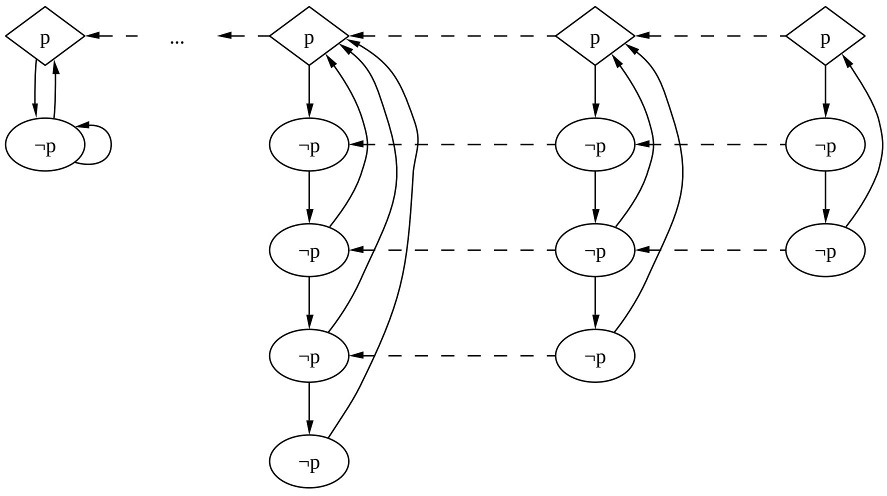

Figure 4.2: A chain of models that refine E and satisfy $\Box\Box\neg p$.

Therefore, the question of stopperedness is important as a guarantee for the existence of minimal models. In this case it would guarantee that there exists a finite model M such that $E \leftarrow M$ (M refines E), $M \models \Box\Box\neg p$ (M satisfies the given property), and for any model N such that $E \leftarrow N \leftarrow M$ and $N \models \Box\Box\neg p$, then $M \leftarrow N$.

In fact, in this example, it is easy to find such a model and, moreover, one that is indeed the *minimum* of $\text{mod}_{\text{FIN}}(\Box\Box\neg p)$ with respect to $\leq_E$. This minimum model can be seen in figure 4.3.

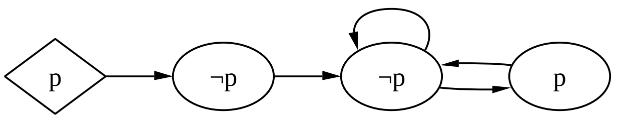

Figure 4.3: M, the minimum of $\text{mod}_{\text{FIN}}(\Box\Box\neg p)$ with respect to $\leq_E$.

We will give an informal proof of the minimality of M. Observe figure 4.4. The fact we are trying to prove is, essentially, that if there exists a model N with all the requirements, and is lower or equal to M then it is equal. More formally, if $N \models \Box\Box\neg p$ and $E \leftarrow N \leftarrow M$ then $N \rightarrow M$. A candidate model for N is shown in the middle of figure 4.4 (for the purposes of readability some of the arrows of the simulation relations have been omitted).

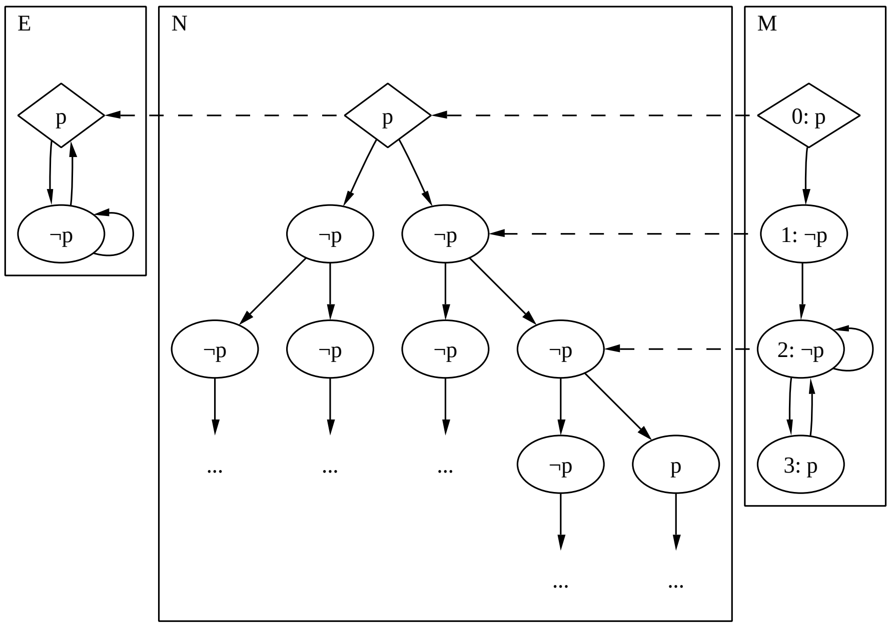

Figure 4.4: E, M and a candidate model N for $\text{mod}_{\text{FIN}}(\Box\Box\neg p)$, $E \leftarrow N \leftarrow M$ and $N \nrightarrow M$.

First, observe that N must have a p-starting state, due to the fact that $N \leftarrow M$. From the same fact it follows that the starting state of N must have at least one $\neg p$-successor. Similarly, from $E \leftarrow N$ it follows that r_N cannot have a p-successor. Also, because of the requirement $N \models \Box\Box\neg p$, there cannot be any states in N with distance 2 from r_N that have p. Therefore, N must look like what is shown in figure 4.4 (modulo simulation equivalence).

Proving that M is the minimum model requires, essentially, the construction of a simulation from N to M. We will briefly define one such relation and argue that it is a simulation. r_N will of course be mapped to r_M. The level-1 and level-2 successors of N will be mapped to states 1 and 2 of M, respectively. For the remaining states, the mapping is defined according to the valuation: a $\neg p$-state of N will be mapped to state 2 and a p-state of N to state 3. The mapping of the $\neg p$ states is trivially adequate, since state 2 has both $\neg p$- and p-successors. The mapping of the p-state is slightly trickier in that state 3 has only a $\neg p$-successor. However, from the requirement that $E \leftarrow N$ follows that a p-state can only transition to a $\neg p$-state. Therefore, the defined relation is a simulation and as such M is the minimum.

This example is obviously a very simple one. We would like to be able to guarantee the existence of minimal models and be able to *compute* them, without recourse to a proof for each individual case. Therefore, it would be interesting to investigate the decidability of the problem of finding minimal models, as well as of reasoning about them.

4.3 M-Saturated models

Minimal refinement from the perspective of m-saturated models, $*_{\mathrm{MSAT}}$, will be discussed in this section. Essentially, an m-saturated model is a Kripke model that satisfies a condition similar to *compactness* across the successors of its states. The formal definition follows.

Definition 4.3.1 (M-saturation)
Let M be a model, s a state in W_M and T a set of sentences.

1. T will be called *satisfiable on the successors of s* iff for each relation R^i_M there exists a state $t \in W_M$ such that $(s, t) \in R^i_M$ and $T \subseteq \mathrm{th}(t)$.

2. Similarly, T will be called *finitely-satisfiable on the successors of s* iff for each relation R^i_M and for any finite set of sentences $F \subseteq T$ there exists an R^i_M-successor t of s such that $F \subseteq \mathrm{th}(t)$.

3. A state s is called *m-saturated* iff, for any set of sentences T, if T is finitely-satisfiable on the successors of s, then it is satisfiable on the successors of s.

4. A model is m-saturated if all its states are m-saturated. □

MSAT will denote the class of m-saturated models. Notice that MSAT is closed under bisimulation.

In the following we will use the *ultrafilter extension* of a model. The internals of the construction are not relevant here, apart from two of its properties:

- the ultrafilter extension of a model M is another model $\mathrm{ue}(M)$ that is logically equivalent to M and,
- $\mathrm{ue}(M)$ is m-saturated.

Accounts of the construction appear in many places, e.g. [BdRV01].

The first two lemmas characterise simulation in syntactic terms, and establish an exact match in the m-saturated case.

Lemma 4.3.2 (Simulation implies PU language inclusion)
If M, N are models such that $M \leftarrow N$, then $\mathrm{PU}(M) \subseteq \mathrm{PU}(N)$ *(Folklore).* □

Lemma 4.3.3 (The effect of PU language inclusion)
Let M, N be models. If $\mathrm{PU}(M) \subseteq \mathrm{PU}(N)$ *and M is m-saturated, then there exists a simulation from N to M, $M \leftarrow N$ (Folklore).* □

Proof For convenience we will work with PE formulae, the dual of positive universal ones. Note that $\mathrm{PU}(s) \subseteq \mathrm{PU}(t)$ iff $\mathrm{PE}(s) \supseteq \mathrm{PE}(t)$. Define a relation S such that

$$(s,t) \in S \quad \text{iff} \quad s \in W_N, t \in W_M \text{ and } \mathrm{PE}(s) \subseteq \mathrm{PE}(t)$$

We prove that S is a simulation.

- Since $\mathcal{A} \cup \overline{\mathcal{A}} \subseteq \mathcal{L}_{\mathrm{PE}}$, S respects the valuations, i.e. if $(s,t) \in S$ then $v_N(s) = v_M(t)$.
- Assume that state s has a successor s' with respect to a relation R^i_N. Let P be the set of PE sentences of s'. For any finite subset $F \subseteq P$, $s' \models \bigwedge F$ and thus $s \models \Diamond_i \bigwedge F$. $\Diamond_i \bigwedge F$ is a PE formula, so by definition it is satisfied at t. Thus there is an R^i_M-successor of t that satisfies $\bigwedge F$. In other words, P is finitely-satisfiable on the successors of t. M however is m-saturated, thus there is an R^i_M-successor t' of t that satisfies P and as such $\mathrm{PE}(s') \subseteq \mathrm{PE}(t')$.

So, S is a simulation whenever it is non-empty and it relates the initial states. Those conditions are satisfied by the assumption $\mathrm{PU}(M) \subseteq \mathrm{PU}(N)$ or equivalently $\mathrm{PE}(N) \subseteq \mathrm{PE}(M)$. ■

A set of sentences T is called *complete* if for every sentence $\phi \in \mathcal{L}_K$, either $T \vdash \phi$ or $T \vdash \neg\phi$. It is easy to see that if T is complete and consistent then there exists a model M such that $\mathrm{th}(M) = T$. But what happens when we restrict to the language of positive universal formulae? The following definition and lemma establish such a criterion.

Definition 4.3.4 (Taking disjuncts and $\mathcal{L}_{\mathrm{PU}}$-consequence)
Let T be a set of sentences.

- T is *closed under taking disjuncts* iff whenever $\phi \vee \psi \in T$ then $\phi \in T$ or $\psi \in T$.
- T is *closed under $\mathcal{L}_{\mathrm{PU}}$-consequence* iff whenever $T \vdash \phi$ and $\phi \in \mathcal{L}_{\mathrm{PU}}$ then $\phi \in T$. □

Lemma 4.3.5 (Exact satisfaction of a set of PU sentences)
Let $P \subseteq \mathcal{L}_{\mathrm{PU}}$. There exists a model M such that $P = \mathrm{PU}(M)$ if and only if P is consistent, closed under $\mathcal{L}_{\mathrm{PU}}$-consequence and taking disjuncts. □

Proof The left-to-right direction is trivial. Right-to-left: for a model M to have exactly P as its set of PU formulae, it must satisfy P and falsify its complement with respect to $\mathcal{L}_{\mathrm{PU}}$. In other words, there exists such a model iff $P, \overline{P^c} \nvdash \bot$. Assume the contrary. Then there exist formulae $\phi, \psi_1, \ldots, \psi_n$ such that $\phi \in P$ (note that P is closed under conjunction), $\neg\psi_i \in \overline{P^c}$ and $\phi, \neg\psi_1, \ldots, \neg\psi_n \vdash \bot$. But then, $\phi \vdash \psi_1 \vee \ldots \vee \psi_n$ and since P is closed under $\mathcal{L}_{\mathrm{PU}}$-consequence, $\psi_1 \vee \ldots \vee \psi_n \in P$. P is also closed under taking disjuncts so there exists $1 \leq j \leq n$ such that $\psi_j \in P$ which is a contradiction because $\psi_j \in P^c$. ■

We may now turn to the first question about minimal refinement in MSAT, non-triviality.

Lemma 4.3.6 (Non-triviality for MSAT)
Let M be an m-saturated model and T a set of sentences. Then,

- *there exists an m-saturated model N such that $M \leftarrow N$ and $N \models T$ iff* $\mathrm{PU}(M) \cup T \nvdash \bot$.
- *there exists an m-saturated model N such that $M \leftarrow N$ and $N \models_G T$ iff* $\mathrm{PU}(M) \cup \Box^* T \nvdash \bot$. □

Proof We consider the local case first. Left-to-right: Since $M \leftarrow N$ it follows from lemma 4.3.2 that $\mathrm{PU}(M) \subseteq \mathrm{PU}(N)$. Thus N is a model of both T and $\mathrm{PU}(M)$.

Right-to-left: Let N be a model of $\mathrm{PU}(M) \cup T$. Then, $\mathrm{PU}(M) \subseteq \mathrm{PU}(N)$. Since N may not be m-saturated, we take the ultrafilter-extension of N, ue(N) which is logically equivalent to N (and as such a model of PU(M) $\cup$ T) and m-saturated. It follows that $T \subseteq$ th(ue(N)) and that PU(M) $\subseteq$ PU(ue(N)). As M is m-saturated it follows from lemma 4.3.3 that $M \leftarrow$ ue(N).

It is easy to see that the condition of non-triviality for the global case follows easily from the local one in view of the observation that $\mathrm{mod}^G_{\mathrm{MSAT}}(T) = \mathrm{mod}_{\mathrm{MSAT}}(\Box^* T)$. ■

The following lemma and theorem concern stopperedness of the ordering for m-saturated models. Lemma 4.3.7 enables us to apply Zorn's lemma by proving that, for any chain[1], a suitable lower bound can be found, and indeed, the infimum.

Lemma 4.3.7 (Infima of restricted chains of m-saturated models) *Let M be an m-saturated model and T a consistent set of sentences of which M is not a model. Let $C \subseteq \mathrm{mod}_{\mathrm{MSAT}}(T)$ be a non-empty chain with respect to $\leq_M$ where all of its members are simulated by M. Then there exists an m-saturated model of T which is the infimum of C (modulo simulation-equivalence).* □

Proof Define $P = \bigcap_{N \in C} \mathrm{PU}(N)$. We will prove that there exists a model L with $\mathrm{PU}(L) = P$ which satisfies T. Since any model N in the chain is simulated by M, $\mathrm{PU}(M) \subseteq \mathrm{PU}(N)$ and therefore $\mathrm{PU}(M) \subseteq P$. Also, for any two models $N, N' \in C$ it will be the case that $\mathrm{PU}(N) \subseteq \mathrm{PU}(N')$ or $\mathrm{PU}(N') \subseteq \mathrm{PU}(N)$. P is obviously consistent as a subset of consistent sets. Also, it is easy to check that P is closed under $\mathcal{L}_{\mathrm{PU}}$-consequence.

We now prove that P is closed under taking disjuncts. Assume $\phi \vee \psi \in P$. Then, for all $N \in C$, $N \models \phi \vee \psi$. If all the models in C satisfy ϕ (or, respectively, ψ) then we are done, so assume that there exists a pair of models $N, N' \in C$ such that $N \models \phi \wedge \neg\psi$ and $N' \models \neg\phi \wedge \psi$. But this contradicts the fact mentioned above, that $\mathrm{PU}(N) \subseteq \mathrm{PU}(N')$ or $\mathrm{PU}(N') \subseteq \mathrm{PU}(N)$. Hence P is closed under taking disjuncts.

From lemma 4.3.5 it follows that $P \cup \overline{P^c}$ is consistent. Assume that $P, \overline{P^c}, T \vdash \bot$. Then there exist formulae $\neg\phi_1, \ldots, \neg\phi_n \in \overline{P^c}$ such that $P, T, \neg\phi_1, \ldots, \neg\phi_n \vdash \bot$ or equivalently $P, T \vdash \phi_1 \vee \ldots \vee \phi_n$. Thus, for all $N \in C$, $N \models \phi_1 \vee \ldots \vee \phi_n$, hence $\phi_1 \vee \ldots \vee \phi_n \in \mathrm{PU}(N)$ and therefore $\phi_1 \vee \ldots \vee \phi_n \in P$. As P is closed under taking disjuncts there is one disjunct ϕ_j such that $\phi_j \in P$, which is a contradiction. Thus there is a model L of $P \cup \overline{P^c} \cup T$. L need not be m-saturated, but its ultrafilter extension ue(L) is, and as it is logically equivalent to L it will satisfy $P \cup \overline{P^c} \cup T$ too.

By the definition of P we have that for all $N \in C$, $\mathrm{PU}(\mathrm{ue}(L)) \subseteq \mathrm{PU}(N)$. Thus, by lemma 4.3.3 we obtain $\mathrm{ue}(L) \leftarrow N$. Also, $\mathrm{PU}(M) \subseteq \mathrm{PU}(\mathrm{ue}(L))$ which implies that $M \leftarrow \mathrm{ue}(L)$. So, ue($L$) is a lower bound of C with respect

[1]A chain is a totally-ordered set of models.

to $\leq_M$. In addition, for any other lower bound L' of C, it follows that $\mathrm{PU}(L') \subseteq \bigcap_{N \in C} \mathrm{PU}(N)$ and thus that $\mathrm{ue}(L)$ is the infimum of C (modulo similarity). ■

In theorems 4.3.10 and 4.4.5 we prove stopperedness for the classes of m-saturated and finite models, respectively. The application of Zorn's lemma is a crucial part of such proofs. The version of this lemma usually found in textbooks is the following:

Lemma 4.3.8 (Zorn's Lemma)
If every non-empty chain of a non-empty partially ordered set X has a lower bound in X, then X possesses a minimal element. □

We will use the following, slightly modified, version of Zorn's lemma:

Lemma 4.3.9 (A slightly stronger version of Zorn's lemma)
If every non-empty chain of a non-empty set X equipped with a preorder *$\leq$ has a lower bound in X, then for any element $x \in X$ there exists an element $y \in X$ such that $y \leq x$ and y is $\leq$-minimal in X.* □

Proof Firstly we define a partial order $\sqsubseteq$ on the basis of $\leq$ as $x \sqsubseteq y$ iff $x < y$ or $x = y$, where $<$ is the strict counterpart of $\leq$ and x, y are members of X. This definition makes $\sqsubseteq$ a partial order as it is transitive, reflexive and antisymmetric.

Assume $x \in X$ and define $X' = \{x' \in X \mid x' \sqsubseteq x\}$. Obviously, X' is non-empty and it is also partially-ordered, as a restriction of a partial order remains a partial order.

Every non-empty chain C of X' with respect to $\sqsubseteq$ is also a non-empty chain of X. Therefore, by assumption, it has a lower bound c within X. But since $C \subseteq X'$, it follows that $c \sqsubseteq x$ and therefore $c \in X'$. The conditions of the original version of Zorn's lemma are satisfied, thus, X' has a minimal element y, with respect to $\sqsubseteq$. Since by its definition, X' is downward-closed, y is minimal in X too.

Now, by y's minimality with respect to $\sqsubseteq$ it follows that there is no $z \in X'$ such that $z \sqsubset y$. Equivalently, through the definition of $\sqsubseteq$ and some algebra, $z < y \wedge y \not< z \wedge y \neq z$. This, in turn, by the necessary antisymmetry and anti-reflexivity of a strict counterpart of an ordering, is equivalent to $z < y$. Therefore, y is minimal with respect to $\leq$ in X'. This completes the proof. ■

In relation to stopperedness, we will consider two collections of sets of m-saturated models,

- $\mathcal{C}^{T}_{\mathrm{MSAT}} = \{\mathrm{mod}_{\mathrm{MSAT}}(S) \mid S \subseteq \mathcal{L}_{\mathrm{K}}\}$
- $\mathcal{C}^{GT}_{\mathrm{MSAT}} = \left\{\mathrm{mod}^{G}_{\mathrm{MSAT}}(S) \,\middle|\, S \subseteq \mathcal{L}_{\mathrm{K}}\right\}$

corresponding to local and global satisfaction respectively (T indicates that we consider sets of sentences instead of formulae).

Theorem 4.3.10 (Stopperedness for MSAT)
Let M be an m-saturated model. The ordering $\leq_M$ is stoppered over both $\mathcal{C}^T_{\mathrm{MSAT}}$ and $\mathcal{C}^{GT}_{\mathrm{MSAT}}$. □

Proof Firstly, we consider $\mathcal{C}^T_{\mathrm{MSAT}}$. Let T be a set of sentences. If $T \subseteq \mathrm{th}(M)$ then M is a *minimum* with respect to $\leq_M$ in $\mathrm{mod}_{\mathrm{MSAT}}(T)$, as well as any other m-saturated model N of T such that $M \leftrightarrows N$. It follows that for any m-saturated model N of T there is an m-saturated model of T, i.e. M, which is minimal and $M \leq_M N$. In the case where $M \notin \mathrm{mod}_{\mathrm{MSAT}}(T)$, it may or may not be the case that $\mathrm{PU}(M) \cup T$ is consistent. If not, then by applying lemma 4.3.6 it follows that there are no models in $\mathrm{mod}_{\mathrm{MSAT}}(T)$ that are simulated by M. Hence, only the third clause of the definition of $\leq_M$ can ever apply, rendering all (simulation-distinct) models in $\mathrm{mod}_{\mathrm{MSAT}}(T)$ incomparable. In this case, for any model $N \in \mathrm{mod}_{\mathrm{MSAT}}(T)$ there is a model N' (namely N itself) such that $N' \leq_M N$, where N' is minimal.

Thus, we assume that $\mathrm{PU}(M) \cup T$ is consistent. Because of the second clause of the definition of the ordering, it is easy to see that in this case the set of minimal elements will be a subset of $\mathrm{mod}_{\mathrm{MSAT}}(\mathrm{PU}(M) \cup T)$. Therefore we restrict our attention to the models in $\mathrm{mod}_{\mathrm{MSAT}}(\mathrm{PU}(M) \cup T)$ which, by virtue of lemma 4.3.6, are all simulated by M.

Then, for any non-empty chain C in $\mathrm{mod}_{\mathrm{MSAT}}(\mathrm{PU}(M) \cup T)$ lemma 4.3.7 applies, yielding a lower bound of C within $\mathrm{mod}_{\mathrm{MSAT}}(\mathrm{PU}(M) \cup T)$. Therefore by Zorn's lemma, for any model $N \in \mathrm{mod}_{\mathrm{MSAT}}(T)$ there exists another model $N' \in \mathrm{mod}_{\mathrm{MSAT}}(T)$ such that N' is minimal and $N' \leq_M N$. Thus $\leq_M$ is stoppered over $\mathcal{C}^T_{\mathrm{MSAT}}$.

For the case of $\mathcal{C}^{GT}_{\mathrm{MSAT}}$ we need only observe that since $\mathrm{mod}^G_{\mathrm{MSAT}}(T) = \mathrm{mod}_{\mathrm{MSAT}}(\Box^* T)$, stopperedness over $\mathcal{C}^{GT}_{\mathrm{MSAT}}$ is reduced to stopperedness over $\mathcal{C}^T_{\mathrm{MSAT}}$. ■

To summarise the set of results on m-saturated models,

- For any set of modal sentences T and an m-saturated model M, the minimal refinement $M *^{[G]}_{\mathrm{MSAT}} T$ is non-trivial (in the sense that the third clause of definition 1.3.1 for $\leq_M$ is *not* the only one that applies within $\mathrm{mod}^{[G]}_{\mathrm{MSAT}}(T)$), if and only if the set $\mathrm{PU}(M) \cup T$ is consistent, in the local case, or $\mathrm{PU}(M) \cup \Box^* T$ in the global case.

- For any m-saturated model M and any set of sentences T,

$$M *^{[G]}_{\mathrm{MSAT}} T = \min_{\leq_M} \left(\mathrm{mod}^{[G]}_{\mathrm{MSAT}}(T) \right) \neq \emptyset$$

This set of results will form the basis of the corresponding ones in the finite case. This will, in general, be achieved by exploiting the fact that finite models are m-saturated, and then by constructing finite models that lie 'below' (in terms of $\leq_M$) the m-saturated ones provided by lemma 4.3.6 and theorem 4.3.10.

4.4 Finite models

We will now turn to the treatment of minimal refinement in the finite case. As already mentioned, the class of all finite Kripke models will be denoted by FIN.

Let M be a model and s, t states in M. A *path* from s to t is a finite sequence of states of M such that the first state is s, the last is t and for any pair of states s_i, s_{i+1} in the sequence, there exists a j such that $(s_i, s_{i+1}) \in R^j_M$. The *depth* of a state s is defined as the minimum length of a path from the root to s if such a path exists, otherwise as ω. s is said to have *in-degree one* whenever it has a unique ancestor with respect to the union of all accessibility relations in M. M will be called *smooth* iff every state in W_M apart from the root has in-degree one and finite depth. For every model M there is a smooth one M^s, known also as the *unfolding* of M, such that $M \sim M^s$. The proof of this result as well as of a general version of the following lemma can be found in [dR95]. This lemma will allow us, in what follows, to concentrate on functional simulations instead of arbitrary ones.

Lemma 4.4.1 (Existence of functional simulations)
Let N and M be models such that N is smooth, M is m-saturated and $M \leftarrow N$. Then there exists a functional simulation from N to M. □

Proof We will define a function $S : W_N \to W_M$ and prove by induction that for any $t \in W_N$, $\text{PE}(t) \subseteq \text{PE}(S(t))$ (S can then be proved to be a simulation in a manner identical to the one presented in lemma 4.3.3). We set $S(r_N) = r_M$. Since $M \leftarrow N$ it follows from lemma 4.3.2 that $\text{PU}(M) \subseteq \text{PU}(N)$ and thus $\text{PU}(S(r_N)) \subseteq \text{PU}(r_N)$, or $\text{PE}(r_N) \subseteq \text{PE}(S(r_N))$.

Assume that S has been defined for all states in N of depth up to $n-1$ and let $t \in W_N$ be a state of depth n. Since N is smooth, t has a uniquely defined ancestor t' with respect to some relation R^i_N. By the inductive hypothesis, $\text{PE}(t') \subseteq \text{PE}(S(t'))$. So, for any finite set of PE sentences $F \subseteq \text{PE}(t)$, it follows that $t' \models \Diamond_i \bigwedge F$, hence $S(t') \models \Diamond_i \bigwedge F$, and as such, there exists a $u \in W_M$ such that $u \models \bigwedge F$ and $(S(t'), u) \in R^i_M$. In other words, $\text{PE}(t)$ is finitely satisfiable on the R^i_M-successors of $S(t')$ which through the m-saturation of M gives us that $\text{PE}(t)$ is satisfiable at a R^i_M-successor u'. We set $S(t) = u'$ and this completes the proof. ■

As explained in the end of the previous section, we plan to use the results on m-saturated models as a basis for the corresponding ones in the finite case. This, in general, boils down to the following scenario: given a finite model M and a sentence $\phi \in \mathcal{L}_K$, we are interested in some property of $M *^{[G]}_{\mathrm{FIN}} \phi$. Since M is a finite model, it is also an m-saturated one, allowing us to apply one of the results of section 4.3, yielding an m-saturated model N such that $M \leftarrow N$ and $N \models_{[G]} \phi$. All that is needed in order to obtain the corresponding result in the finite case is to construct a finite model L such that $M \leftarrow L \leftarrow N$ and $L \models_{[G]} \phi$. The following definition and lemma will give us exactly such a tool.

Definition 4.4.2 (A modified filtration)
Let Σ be a set of sentences such that $\mathcal{A} \subseteq \Sigma$, M a model and E an equivalence relation on W_M.

1. Σ is *subformula-closed* iff for every ϕ,

   - $\neg\phi \in \Sigma$ implies $\phi \in \Sigma$,
   - $\phi \wedge \psi \in \Sigma$ implies $\phi \in \Sigma$ and $\psi \in \Sigma$ and
   - $\Diamond_i \phi \in \Sigma$ implies $\phi \in \Sigma$, for all $1 \leq i \leq m$.

   (in the unabbreviated language).

2. We define an equivalence relation $\equiv_{\Sigma,E}$ on W_M as follows.

   $$s \equiv_{\Sigma,E} t \quad \text{iff} \quad (\forall \phi \in \Sigma, s \models \phi \Leftrightarrow t \models \phi) \wedge (s,t) \in E$$

   We denote the $\equiv_{\Sigma,E}$-equivalence class of s by $[s]$.

3. We define a model M_f as follows

   - $W_{M_f} = \{[s] \mid s \in W_M\}$
   - $r_{M_f} = [r_M]$
   - For all $s, t \in W_M$, $([s],[t]) \in R^i_{M_f}$ if there exist states $s', t' \in W_M$ such that $s' \in [s], t' \in [t]$ and $(s', t') \in R^i_M$.
   - $p \in v_{M_f}([s])$ iff $p \in v_M(s)$ for all $p \in \mathcal{A} \cap \Sigma$. □

It is not hard to see that if Σ is subformula-closed then the model M_f defined above is a slightly modified filtration of M (see, e.g. [Che80, BdRV01]). As such, it has similar properties to a filtration:

- Let n be the number of E-equivalence classes in M. If n and $|\Sigma|$ are finite, then M_f is finite and $|M_f| \leq 2^{|\Sigma|} \cdot n$.

- For all formulae $\psi \in \Sigma$ and states $s \in W_M$, $M, s \models \psi$ iff $M_f, [s] \models \psi$.

Lemma 4.4.3 (Bounded-size intermediary models for $*_{\mathbf{FIN}}$)
Let M be a finite model, L a possibly infinite model such that $M \leftarrow L$ and ϕ a formula. Then there exists a finite model N such that $M \leftarrow N \leftarrow L$ and that $N \models_{[G]} \phi$ iff $L \models_{[G]} \phi$. Moreover, $|N| \leq 2^{|\phi|+|\mathcal{A}|} \cdot |M|$. □

Proof Define Σ as the set of sub-formulae of ϕ unioned with the set of atomic propositions $\mathcal{A}$, along with its atoms negated $\mathcal{A}^c$. Let K be the smooth counterpart of L. Since $L \sim K$ and $M \leftarrow L$ it follows that $M \leftarrow K$. Moreover, since M is finite it is also m-saturated, thus lemma 4.4.1 applies, giving us a functional simulation S between K and M. By $S(t)$ we denote the (unique) state $t' \in W_M$ such that $(t, t') \in S$ for some state $t \in W_K$. Using S we define an equivalence relation E on W_K as

$$(s,t) \in E \quad \text{iff} \quad s,t \in W_K \quad \text{and} \quad S(s) = S(t)$$

Using K, Σ and E we define K_f (which we will abbreviate as N) and the associated $[\cdot]$ mapping. Thus, it follows that

- Σ is finite by definition and has a size of at most $|\phi|+|\mathcal{A}|$. Since S maps K into M, a finite model, the number of E-equivalence classes must be finite and at most $|M|$. Therefore, N is finite and $|N| \leq 2^{|\phi|+|\mathcal{A}|} \cdot |M|$.
- N is a filtration of K and as such the usual properties hold, i.e. $N \models_{[G]} \phi$ iff $K \models_{[G]} \phi$ iff $L \models_{[G]} \phi$ (for the global case, note that $[\cdot]$ is surjective on W_N).

We now prove that the necessary simulations exist between M, N, L.

$N \leftarrow L$: We first prove that $N \leftarrow K$. Since $L \sim K$ it then follows that $N \leftarrow L$ as well. Define a relation $Q \subseteq W_K \times W_N$ such that

$$(s,t) \in Q \quad \text{iff} \quad t = [s]$$

We prove that Q is a simulation:

- Obviously, $(r_K, r_N) \in Q$ since $r_N = [r_K]$.
- Also, $(s, [s]) \in Q$ implies $v_K(s) = v_N([s])$ by the definition of v_K and from the fact that Σ contains all the propositional letters.
- If $(s,t) \in R^i_M$ then by definition there is an R^i_N-successor of $[s]$ that is the image of t, namely $[t]$.

$M \leftarrow N$: Define a relation $U \subseteq W_N \times W_M$ as

$$(t,u) \in U \quad \text{iff} \quad \exists s \in W_K,\ [s] = t \wedge S(s) = u$$

We prove that U is a simulation.

- By definition, $[r_K] = r_N$. Since S is a simulation, $S(r_K) = r_M$. Thus, $(r_N, r_M) \in S$.
- Suppose that $(t, u) \in S$. Thus, there exists a state s in K such that $[s] = t$, therefore $v_K(s) = v_N(t)$ given that $\mathcal{A} \subseteq \Sigma$. Moreover, $S(s) = u$, thus $v_K(s) = v_M(u)$. Therefore, $v_N(t) = v_M(u)$.
- Assume that $(t_1, t_2) \in R^i_N$. That means that there exist two states s_1, s_2 in K such that $[s_1] = t_1, [s_2] = t_2$ and $(s_1, s_2) \in R^i_K$. As such, since S is a simulation, $(S(s_1), S(s_2)) \in R^i_M$. Thus, $(t_1, S(s_1)) \in U$ and $(t_2, S(s_2)) \in U$. ■

With the help of this result we may approach the questions on minimal refinement in the finite case. We start from non-triviality.

Lemma 4.4.4 (Non-triviality for FIN)
Let ϕ be a formula and M a finite model. Then,

- *there exists a finite model N such that $N \models \phi$ and $M \leftarrow N$ iff* $\mathrm{PU}(M), \phi \nvdash \bot$.
- *there exists a finite model N such that $N \models_G \phi$ and $M \leftarrow N$ iff* $\mathrm{PU}(M) \cup \Box^*\phi \nvdash \bot$.

Moreover, it is decidable whether such a model exists. □

Proof Left-to-right: In the local case, $N \models \phi$. Since $M \leftarrow N$, it follows that $\mathrm{PU}(M) \subseteq \mathrm{PU}(N)$ so it cannot possibly be the case that $\mathrm{PU}(M)$ is inconsistent with ϕ. The global case follows similarly by observing that $N \models_G \phi$ iff $N \models \Box^*\phi$.

Right-to-left: We address the local and global case simultaneously. Since $\mathrm{PU}(M) \cup \phi$ is consistent ($\mathrm{PU}(M) \cup \Box^*\phi$), from lemma 4.3.6 it follows that there exists an m-saturated model N' such that $N' \models \phi$ ($N' \models \Box^*\phi$ or equivalently $N' \models_G \phi$), and $M \leftarrow N'$. But then, by lemma 4.4.3 there exists a finite model $N \models \phi$ ($N \models_G \phi$) such that $M \leftarrow N \leftarrow N'$ and $|N| \leq |M| \cdot 2^{|\phi| + |\mathcal{A}|}$.

Since the size of N is bounded we can enumerate all the models that satisfy (globally satisfy) ϕ with up to $|M| \cdot 2^{|\phi|+|\mathcal{A}|}$ states and check whether any of these is simulated by M. Thus the problem is decidable. ■

As with m-saturated models, we will again consider two collections of sets of finite models for stopperedness:

- $\mathcal{C}^{\phi}_{\mathrm{FIN}} = \{\mathrm{mod}_{\mathrm{FIN}}(\phi) \mid \phi \in \mathcal{L}_{\mathrm{K}}\}$
- $\mathcal{C}^{G\phi}_{\mathrm{FIN}} = \left\{\mathrm{mod}^{G}_{\mathrm{FIN}}(\phi) \mid \phi \in \mathcal{L}_{\mathrm{K}}\right\}$

Theorem 4.4.5 (Stopperedness for FIN)
Let M be a finite model. The ordering $\leq_M$ is stoppered over C^{ϕ}_{FIN} and $C^{G\phi}_{\text{FIN}}$. □

Proof As in the proof of theorem 4.3.10, it is easy to check that when $M \models_{[G]} \phi$ or $\text{PU}(M), \phi \vdash \bot$ ($\text{PU}(M) \cup \Box^* \phi \vdash \bot$) then for any model $N \in \text{mod}^{[G]}_{\text{FIN}}(\phi)$ there exists a model $N' \in \text{mod}^{[G]}_{\text{FIN}}(\phi)$ such that $N' \leq_M N$ and N' is minimal. So we assume that $M \not\models_{[G]} \phi$, that $\text{PU}(M), \phi \not\vdash \bot$ ($\text{PU}(M) \cup \Box^* \phi \not\vdash \bot$) and restrict our attention to the models in $\text{mod}_{\text{FIN}}(\text{PU}(M) \cup \{\phi\})$ ($\text{mod}_{\text{FIN}}(\text{PU}(M) \cup \Box^* \phi)$).

Let $C \subseteq \text{mod}_{\text{FIN}}(\text{PU}(M) \cup \{\phi\})$ ($C \subseteq \text{mod}_{\text{FIN}}(\text{PU}(M) \cup \Box^* \phi)$) be a chain with respect to $\leq_M$. Since finite models are m-saturated, from theorem 4.3.7 we obtain that there is an m-saturated model L which satisfies $\text{PU}(M) \cup \{\phi\}$ ($\text{PU}(M) \cup \Box^* \phi$) and is a lower bound of C with respect to $\leq_M$. But then, by lemma 4.4.3, there is a finite model N such that $N \models_{[G]} \phi$ and $M \leftarrow N \leftarrow L$. Therefore, N is a lower bound of C and by applying Zorn's lemma we obtain stopperedness for the class of finite models. ■

We continue with a set of decidability results concerning the finite case. Firstly we prove that checking whether a specific model N is minimal with respect to a model M and ϕ, i.e. whether $N \in M *^{[G]}_{\text{FIN}} \phi$, is decidable.

Lemma 4.4.6 (Decidability of minimality)
Let M, N be finite models and ϕ a sentence. There is an effective procedure for deciding $N \in M *^{[G]}_{\text{FIN}} \phi$. □

Proof We first test for non-triviality via lemma 4.4.4. If $M *^{[G]}_{\text{FIN}} \phi$ is trivial then we check whether $N \models_{[G]} \phi$. In this case, N is minimal iff $N \models_{[G]} \phi$.

Thus, we assume that $M *^{[G]}_{\text{FIN}} \phi$ is non-trivial. Then, we check whether $M \leftarrow N$. If not, then surely N is not minimal. So, we assume that $M \leftarrow N$. Now, N is minimal iff there is no other model $N' \in \text{mod}^{[G]}_{\text{FIN}}(\phi)$ such that $N' <_M N$. Assume N is not minimal. Then there exists $N' \in \text{mod}^{[G]}_{\text{FIN}}(\phi)$ such that $M \leftarrow N' \leftarrow N$ but $N' \not\twoheadrightarrow N$ (by the definition of the ordering and the assumption that $M \leftarrow N$ it follows that if there exists $N' <_M N$, it will have to be due to the first clause of the definition). By applying lemma 4.4.3 to the pair of models M, N' it follows that there exists a model $N'' \in \text{mod}^{[G]}_{\text{FIN}}(\phi)$ such that $M \leftarrow N'' \leftarrow N'$ and thus, $N'' \not\twoheadrightarrow N$. Therefore, N is not minimal iff there exists a model N'' of ϕ which is strictly smaller than N with respect to $\leq_M$ and it has at most $|M| \cdot 2^{|\phi|+|\mathcal{A}|}$ states.

Consequently, given N, M and ϕ, we can enumerate all the models that satisfy ϕ (globally satisfy) have up to $|M| \cdot 2^{|\phi|+|\mathcal{A}|}$ states, of which there is a finite number. For each model L we check the simulations $M \leftarrow L$, $L \leftarrow N$, $L \not\twoheadrightarrow N$. There exists such a model iff $N \notin M *^{[G]}_{\text{FIN}} \phi$. ■

The next theorem characterises the structure of $M *_{\text{FIN}}^{[G]} \phi$ with respect to the ordering. It asserts that each $\leq_M$-equivalence class of models in $M *_{\text{FIN}}^{[G]} \phi$ contains a representative model of a bounded size.

Theorem 4.4.7 (Computable representatives of $*_{\text{FIN}}$)
Let M be a finite model and ϕ a formula such that $M *_{\text{FIN}}^{[G]} \phi$ is non-trivial. Then, there is a finite, computable set of finite models $\Delta_{M,\phi}^{[G]} \subseteq M *_{\text{FIN}}^{[G]} \phi$ with the property that for any model $N \in M *_{\text{FIN}}^{[G]} \phi$ there is a model N' such that

- $N' \in \Delta_{M,\phi}^{[G]}$,
- $N \leftrightarrows N'$ *and*
- $|N'| \leq |M| \cdot 2^{|\phi|+|\mathcal{A}|}$. □

Proof Let $N \in M *_{\text{FIN}}^{[G]} \phi$. The application of lemma 4.4.3 gives us a model N' of ϕ such that $M \leftarrow N' \leftarrow N$ and $|N'| \leq |M| \cdot 2^{|\phi|+|\mathcal{A}|}$. But since N is minimal, it follows that $N \leftrightarrows N'$. Thus $\Delta_{M,\phi}^{[G]}$ can be computed by enumerating the finite models of ϕ that have at most $|M| \cdot 2^{|\phi|+|\mathcal{A}|}$ states and checking them for minimality via lemma 4.4.6. ■

A corollary of the above is that if for some finite model M and a formula ϕ, $M *_{\text{FIN}}^{[G]} \phi$ is non-trivial, then the number of simulation-equivalence classes of finite models that constitute $M *_{\text{FIN}}^{[G]} \phi$ is finite. We will now examine the decidability of reasoning about the results of the operation.

Theorem 4.4.8 (Decidability of reasoning about minimal models)
Assume that $M *_{\text{FIN}}^{[G]} \phi$ is non-trivial. Let ψ be a formula in $\mathcal{L}_{\text{PU}} \cup \mathcal{L}_{\text{PE}}$. Then, there is an effective procedure for deciding $M *_{\text{FIN}}^{[G]} \phi \models_{[G]} \psi$ (the global and local cases can be combined, i.e. $M *_{\text{FIN}} \phi \models_G \psi$). □

Proof Consider the partitioning of $M *_{\text{FIN}}^{[G]} \phi$ by simulation-equivalence: in view of the previous result, there is a (finite) number of simulation-equivalence classes of finite models in $M *_{\text{FIN}}^{[G]} \phi$. Let $E \subseteq M *_{\text{FIN}}^{[G]} \phi$ be such an equivalence class. For any two models $N_1, N_2 \in E$ it holds that $\text{PU}(N_1) = \text{PU}(N_2)$ and equivalently $\text{PE}(N_1) = \text{PE}(N_2)$. In other words, the problem of checking whether all models in $M *_{\text{FIN}}^{[G]} \phi$ satisfy ψ (locally/globally), where $\psi \in \mathcal{L}_{\text{PU}} \cup \mathcal{L}_{\text{PE}}$, reduces to checking whether for every simulation-equivalence class E in $M *_{\text{FIN}}^{[G]} \phi$, there is a model $N \in E$ such that $N \models_{[G]} \psi$. But $\Delta_{M,\phi}^{[G]}$ contains at least one model from each such equivalence class, so the problem is further reduced to whether $\Delta_{M,\phi}^{[G]} \models_{[G]} \psi$ or not. Since $\Delta_{M,\phi}^{[G]}$ is finite and computable, and the problem of checking (global/local) satisfaction of a formula on a finite model is decidable, the overall problem is also decidable. ■

Indeed, it is easy to see that the query language can be any language that is preserved under simulation-equivalence, e.g. ACTL.

4.5 Related work

Van der Meyden, in [vdM91], describes a framework designed to support artificial intelligence models of legal reasoning. Specifically, a logic for deontic action specification is developed, and proved to be complete with respect to the semantics presented therein. This logic is related to the framework presented here in that a process of minimisation is central to the model theory of the language, and that part of the ordering relevant to this minimisation is simulation. However, the approaches still differ significantly. Firstly, the work presented here is geared towards reasoning about specifications and processes rather than a framework designed for AI reasoning. Secondly, van der Meyden's work is closer to logic programming than this research, in that the minimisation is employed to provide unique models of theories (minima), rather than to perform changes to a theory or a model. Indeed, van der Meyden's effort is concentrated in ensuring a complete proof theory with which derivation can be performed, rather than using the models as possible representations of systems. Lastly, the language considered is a clausal one, i.e. one that only allows reasoning about rules that contain negation only in their heads, and not in their bodies.

Chapter 5

Minimal refinement and Temporal Logic

5.1 Introduction

Modal logic is a basic language for expressing specifications and describing processes. However, it is limited in what kinds of specifications it can express. For example, consider the case whereby a designer wants to minimally refine a system by a property of the form 'in all possible states of the system, if the atomic variable Request is true then eventually RequestGranted becomes true'. This property dictates that for every state s that satisfies Request, there is a finite path i starting at s and ending at a state t, such that $t \models \mathsf{RequestGranted}$. It is easy to see that a modal formula ϕ cannot express such a property as it can only impose restrictions on states whose distance from s is at most as great as the nesting depth of the modalities in ϕ; since that depth is finite, restrictions on states whose distance from s is greater than that depth cannot be enforced and therefore an artificial bound is put on the time the variable RequestGranted is satisfied.

To address this lack of expressiveness we will instantiate the framework of minimal refinement for a temporal logic. We chose the logic ACTL [GL94], a fragment of CTL [CES86], a branching-time temporal logic well-known for its use in model checking. ACTL is a definite improvement on modal logic, since it provides temporal operators such as *until* and *eventually*. It is also incomparable with modal logic, since it is only universally quantified; for example, the modal property 'there is a next state that makes the atomic variable p true' cannot be expressed in ACTL. Finally, ACTL is an expressive language that is preserved by (fair) simulation, a fact that leads to a set of results on minimal refinement obtained in a straightforward way.

In the following we will also do away with the distinction between local and global minimal refinement; a formula of the form *globally-ϕ* (AGϕ) immediately implies the satisfaction of ϕ on all of the (reachable) states of a model, rendering unnecessary the existence of two versions of the operation.

The outline of this chapter is as follows. Minimal refinement with ACTL as the object language and transition systems with fairness constraints is examined in section 5.2. The issues of non-triviality, stopperedness and decidability are discussed in the same section. Finally, an implementation is developed and a case study examined in section 5.3.

5.2 Minimal refinement in the temporal case

In this chapter, *fair simulation* is used to instantiate the notion of refinement. Thus, the ordering $\leq_M$ will now be understood to involve fair simulations where it makes use of refinements ($\leftarrow$). The operation of minimal refinement in this context is

$$M *_{\mathrm{FTSF}} \phi = \min_{\leq_M}(\mathrm{mod}_{\mathrm{FTSF}}(\phi))$$

where $M \in \mathcal{M}_{\mathrm{FTSF}}$ and $\phi \in \mathcal{L}_{\mathrm{ACTL}}$.

Of course, the problem of triviality applies in this case too. It is easy to see that if a ACTL formula does not force an inconsistent initial state (e.g. a formula like $p \wedge \neg p$), then it has a very simple model: one that only contains a few initial states and no transitions at all. It is also easy to verify that such a model refines any other model that is compatible with it, in terms of its initial states valuation (note that such a model satisfies almost all ACTL formulae: the ones it does not are of the form $\phi_p \wedge \psi$ where ϕ_p is a propositional formula which is false on its initial states). The next lemma formalises this intuition, which incidentally demonstrates that in the context of ACTL, non-triviality is a less interesting issue. For the purposes of the next lemma, a formula that is formed by propositional atoms in some set $\mathcal{A}$, and the connectives $\neg, \wedge, \vee$, will be called a *propositional formula on* $\mathcal{A}$.

Lemma 5.2.1 (Non-triviality for ACTL)
Let M be a model in FTSF *and ϕ a satisfiable formula of* ACTL*. Then, the minimal refinement $M *_{\mathrm{FTSF}} \phi$ is non-trivial iff there exists a state $s \in S_M$ such that for any propositional formula ψ on $\mathcal{A}_M$, $\phi \models \psi$ implies that $M, s \models \psi$.* □

Proof Left-to-right: Assume that $M *_{\mathrm{FTSF}} \phi$ is non-trivial; then, there exists a model N such that $N \models \phi$ and $M \leftarrow N$. So, pick any state $t \in S_N$ and its image $s \in S_M$ under the witnessing fair simulation for $M \leftarrow N$. Also, pick any propositional formula ψ on $\mathcal{A}_M$, which by the fact that $\mathcal{A}_M \subseteq \mathcal{A}_N$ (by definition 2.4.4 and $M \leftarrow N$), is a propositional formula on $\mathcal{A}_N$. If $\phi \models \psi$ then $N \models \psi$ (because $N \models \phi$), i.e., for all states $t' \in S_N$, $N, t' \models \psi$ and therefore $N, t \models \psi$. But ψ is also a ACTL formula, and as such, it is preserved by fair simulation. Therefore, $M, s \models \psi$.

Right-to-left: Given $s \in S_M$ with the required assumptions, define a model N such that $W_N = S_N = \{t\}$, $R_N = \mathcal{F}_N = \emptyset$, $\mathcal{A}_N = \mathcal{A}_M$ and $v_N(t) = v_M(s)$. It is easy to see that, trivially, $M \leftarrow N$. Now, assume that $N, t \not\models \phi$. Because of the absence of any fair path in N starting at T, it must be that the valuation of t is the one that falsifies ϕ, something contradictory with the assumption. ∎

We will proceed with a few lemmas that will form the basis for the rest of the results on $*_{\mathrm{FTSF}}$. The definition of a product of two models in $\mathcal{M}_{\mathrm{FTSF}}$ follows, and is similar to the natural one, plus the item that defines the fairness constraints.

Definition 5.2.2 (The synchronous product construction)
Given two models A, B define the *synchronous product* $A \times B$ to be a model with the following parts.

- $W_{A\times B} = \{(a,b) \mid a \in W_A,\ b \in W_B,\ v_A(a) \cap \mathcal{A}_B = v_B(b) \cap \mathcal{A}_A\}$
- $S_{A\times B} = (S_A \times S_B) \cap W_{A\times B}$
- $\mathcal{A}_{A\times B} = \mathcal{A}_A \cup \mathcal{A}_B$
- $v_{A\times B}((a,b)) = v_A(a) \cup v_B(b)$
- $(a,b) \rightarrow_{A\times B} (a',b')$ iff $a \rightarrow_A a'$ and $b \rightarrow_B b'$.
- $\mathcal{F}_{A\times B} = \{(P \times W_B) \cap W_{A\times B} \mid P \in \mathcal{F}_A\} \cup$
  $\{(W_A \times Q) \cap W_{A\times B} \mid Q \in \mathcal{F}_B\}$ □

As noted, the definition of the product structure is the natural one: by matching the common parts of the valuation, we construct composite states. Accordingly, the transition relation is the restriction of the cartesian product of the transition relations of the factors, on the resulting state space of the product. As expected, it is easy to prove that $A \leftarrow A \times B$ and $B \leftarrow A \times B$ for any A and B. The following result, proved in [GL94], asserts that a sequence of states is a fair path in the product if and only if its projections are fair paths in the product's factors, and the corresponding pairs of states are states in the product.

Lemma 5.2.3 (Fairness of paths in product structures [GL94])
Let A, B be models. The following conditions are equivalent.

1. $\pi_{A\times B} = (a_0,b_0)(a_1,b_1)\ldots$ *is a fair path in* $A \times B$.
2. $\pi_A = a_0a_1\ldots$ *and* $\pi_B = b_0b_1\ldots$ *are fair paths in* A, B *respectively and for all* i, $(a_i,b_i) \in W_{A\times B}$. □

The next lemma asserts that if a model is simulated by both of the factors of a product, then it is simulated by the product as well.

Lemma 5.2.4 (Factors and products)
Let A, B, C be models such that $B \leftarrow A$, $C \leftarrow A$. Then, $B \times C \leftarrow A$. □

Proof Let H_B, H_C be the witnessing fair simulations from A to B, C respectively. Define a relation $H_{B\times C} \subseteq W_A \times W_{B\times C}$,

$$(a,(b,c)) \in H_{B\times C} \quad \text{iff} \quad a \in W_A, (b,c) \in W_{B\times C}, (a,b) \in H_B, (a,c) \in H_C$$

We will prove that $H_{B\times C}$ is a fair simulation from A to $B \times C$. First observe that by assumption $\mathcal{A}_A \supseteq \mathcal{A}_B$ and $\mathcal{A}_A \supseteq \mathcal{A}_C$ so $\mathcal{A}_A \supseteq \mathcal{A}_B \cup \mathcal{A}_C = \mathcal{A}_{A\times B}$.

1. For any state $\alpha \in S_A$ there is a state $(\beta,\gamma) \in S_{B\times C}$ such that $(\alpha,(\beta,\gamma)) \in H_{B\times C}$:

   From the fact that H_B, H_C are fair simulations it follows that there exist states $\beta \in S_B$ and $\gamma \in S_C$ such that $(\alpha,\beta) \in H_B$ and $(\alpha,\gamma) \in H_C$. From the same fact it follows that $v_A(\alpha) \cap \mathcal{A}_B = v_B(\beta)$ and $v_A(\alpha) \cap \mathcal{A}_C = v_C(\gamma)$ so $v_B(\beta) \cap \mathcal{A}_C = v_C(\gamma) \cap \mathcal{A}_B$ and as such $(\beta,\gamma) \in W_{B\times C}$. Moreover, $(\beta,\gamma) \in S_{B\times C}$. From the definition of $H_{B\times C}$ it follows that $(\alpha,(\beta,\gamma)) \in H_{B\times C}$.

2. For all $a \in W_A$ and $(b,c) \in W_{B\times C}$, $(a,(b,c)) \in H_{B\times C}$ implies that

   (a) $v_A(a) \cap \mathcal{A}_{B\times C} = v_{B\times C}(b,c)$:
   As in the previous item, from $(a,(b,c)) \in H_{B\times C}$ we get that $v_A(a) \cap \mathcal{A}_B = v_B(b)$ and $v_A(a) \cap \mathcal{A}_C = v_C(c)$. Thus $v_A(a) \cap \mathcal{A}_{B\times C} = v_A(a) \cap (\mathcal{A}_B \cup \mathcal{A}_C) = (v_A(a) \cap \mathcal{A}_B) \cup (v_A(a) \cap \mathcal{A}_C) = v_B(b) \cup v_C(c) = v_{B\times C}(b,c)$.

   (b) For every fair path $\pi_A = a_0 a_1 \ldots$ in A with $a_0 = a$ there exists a fair path $\pi_{B\times C} = (b_0,c_0)(b_1,c_1)\ldots$ in $B \times C$ with $(b_0,c_0) = (b,c)$ such that for every i, $(a_i,(b_i,c_i)) \in H_{B\times C}$:
   From the fact that H_B, H_C are fair simulations we obtain that there exist fair paths $\pi_B = b_0, b_1, \ldots$ and $\pi_C = c_0, c_1, \ldots$ in B, C respectively, such that $b_0 = b$, $c_0 = c$ and for all i, $(a_i,b_i) \in H_B$ and $(a_i,c_i) \in H_C$.
   Since $(a_i,b_i) \in H_B$ and $(a_i,c_i) \in H_C$, it follows that $v_A(a_i) \cap \mathcal{A}_B = v_B(b_i)$ and $v_A(a_i) \cap \mathcal{A}_C = v_C(c_i)$, thus $v_B(b_i) \cap \mathcal{A}_C = v_C(c_i) \cap \mathcal{A}_B$. As such, $(b_i,c_i) \in W_{B\times C}$. From lemma 5.2.3 it follows that $\pi_{B\times C}$ is a fair path in $B \times C$. Moreover, from the definition of $H_{B\times C}$ it follows that $(a_i,(b_i,c_i)) \in H_{B\times C}$. ■

With this result in our disposal we can aim to directly construct an algorithm and a proof for the stopperedness of $\leq_M$ over $*_{\mathrm{FTSF}}$. The idea is that, since fair simulation preserves ACTL, we can use the product operation to produce a model that refines a designated model M, and also refines another model that satisfies a formula ϕ. Then, the product would have to satisfy ϕ by construction. This last model must, essentially, contain exactly all the possible behaviours allowed by the formula ϕ, in other words, to be a tableau of that formula. Several researchers have investigated such constructions for ACTL, e.g. [GL94, CGL96, Mai00, PMT02] with the following general properties:

Lemma 5.2.5 (Existence of tableau constructions)
A function τ of the type $\mathcal{L}_{\text{ACTL}} \to \mathcal{M}_{\text{FTSF}}$ can be defined such that:

$$\text{for all } M \in \mathcal{M}_{\text{FTSF}}, \tau(\phi) \leftarrow M \text{ iff } M \models \phi$$

Moreover, τ is computable and $|\tau(\phi)| \leq 2^{|\phi|}$. A corollary of the above is that

$$\phi \models \psi \quad \text{iff} \quad \tau(\phi) \models \psi \qquad \square$$

The rest of the results in this section make use of tableau constructions, but by only making reference to their general properties mentioned above. A specific tableau construction, the one described in [CGL96], will be implemented and used in section 5.3. We can now state the main result of this section. The following theorem forms the basis of the results concerning stopperedness and decidability for minimal refinement in ACTL.

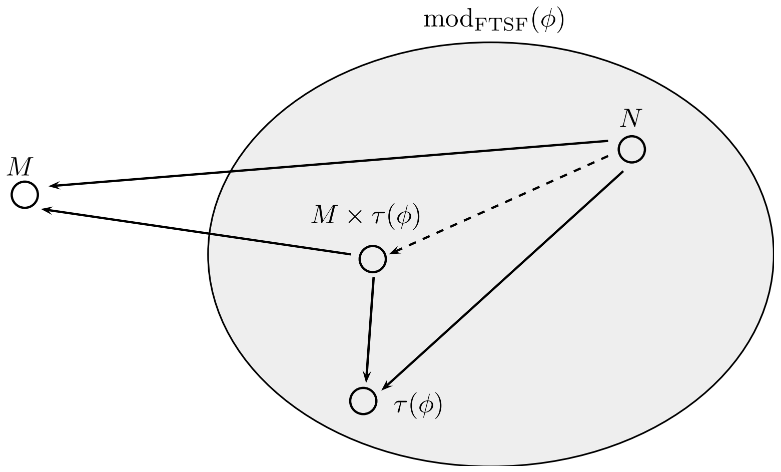

Figure 5.1: $M \times \tau(\phi)$ as the minimal model of ϕ with respect to $\leq_M$.

Theorem 5.2.6 (A representative model of $M *_{\text{FTSF}} \phi$)
For any model N inside $M *_{\text{FTSF}} \phi$, $N \leftrightarrows \tau(\phi) \times M$. $\square$

Proof We prove that among the models of ϕ that are simulated by M, $\tau(\phi) \times M$ is the minimum, modulo fair simulation. Let N be a model of ϕ.

Assume that $M \not\leftarrow N$. In that case it follows trivially by the second clause of the definition of $\leq_M$ that $\tau(\phi) \times M \leq_M N$, since $M \leftarrow \tau(\phi) \times M$ as mentioned previously.

Thus, we assume that $M \leftarrow N$ (see figure 5.1). Since N is a model of ϕ, it must be the case that $\tau(\phi) \leftarrow N$, because $\tau(\phi)$ is the tableau of ϕ. Therefore, $\tau(\phi) \times M \leftarrow N$ by applying lemma 5.2.4. Also, it holds that $M \leftarrow \tau(\phi) \times M$, thus $\tau(\phi) \times M \leq_M N$.

Thus, for all $N \in \mathrm{mod}_{\mathrm{FTSF}}(\phi)$, it holds that $\tau(\phi) \times M \leq_M N$. By the preservation of ACTL by fair simulation if follows that $\tau(\phi) \times M$ is a model of ϕ and therefore is the minimum with respect to $\leq_M$ in $\mathrm{mod}_{\mathrm{FTSF}}(\phi)$. ■

The above result answers the questions of stopperedness and decidability in one go. Specifically, the ordering $\leq_M$ is obviously stoppered over

$$\mathcal{C}_{\mathrm{FTSF}} = \{\mathrm{mod}_{\mathrm{FTSF}}(\phi) \mid \phi \in \mathcal{L}_{\mathrm{ACTL}}\}$$

Moreover, since $\tau(\phi) \times M$ is finite and computable it follows that there is an effective procedure for checking $N \in M *_{\mathrm{FTSF}} \phi$, i.e. checking whether $N \leftrightarrows \tau(\phi) \times M$. Finally, for any $\psi \in \mathcal{L}_{\mathrm{ACTL}}$ there is an effective procedure for checking $M *_{\mathrm{FTSF}} \phi \models \psi$, i.e. checking whether $\tau(\phi) \times M \models \psi$ (observe that since ACTL is preserved by fair simulation there is no need to restrict ψ as was the case with modal logic).

Finally, we present a simple result that will be used later, in section 5.3. Essentially, it states that the tableau of a conjunction can be viewed as the product of the tableaus of the conjuncts. This, in turn, characterises iteration over minimal refinements in ACTL: a sequence of minimal refinements is equivalent to a single minimal refinement by the conjunction of the formulae appearing in the sequence.

Lemma 5.2.7 (Tableaus and conjunctions)
For all $\phi, \psi \in \mathcal{L}_{\mathrm{ACTL}}$, $\tau(\phi \wedge \psi) \leftrightarrows \tau(\phi) \times \tau(\psi)$. Therefore, $M *_{\mathrm{FTSF}} \phi *_{\mathrm{FTSF}} \psi \leftrightarrows M *_{\mathrm{FTSF}} \phi \wedge \psi$. □

Proof By the properties of the product, it follows that $\tau(\phi) \leftarrow \tau(\phi) \times \tau(\psi)$. Fair simulation preserves ACTL, thus, since $\tau(\phi) \models \phi$ it follows that $\tau(\phi) \times \tau(\psi) \models \phi$ and similarly $\tau(\phi) \times \tau(\psi) \models \psi$ therefore $\tau(\phi) \times \tau(\psi) \models \phi \wedge \psi$. But then, by lemma 5.2.5 we get that $\tau(\phi \wedge \psi) \leftarrow \tau(\phi) \times \tau(\psi)$.

Obviously $\tau(\phi \wedge \psi) \models \phi \wedge \psi$ therefore $\tau(\phi \wedge \psi) \models \phi$. By applying again lemma 5.2.5 we obtain $\tau(\phi) \leftarrow \tau(\phi \wedge \psi)$ and similarly $\tau(\psi) \leftarrow \tau(\phi \wedge \psi)$. With the help of lemma 5.2.4 we are led to $\tau(\phi) \times \tau(\psi) \leftarrow \tau(\phi \wedge \psi)$. This completes the proof. ■

5.3 Implementation and case study

We have proceeded to develop a prototype implementation of minimal refinement for ACTL and FTSF. It is easy to see that the product operation is identical to the synchronous composition operation found in branching-time model checkers like SMV [McM93]. Moreover, since SMV provides an already existing framework for CTL (and thus ACTL) model-checking we decided to use it for the purpose of (implicitly) computing the product structure $\tau(\phi) \times M$, after having generated $\tau(\phi)$ with our implementation.

Therefore, our implementation must be able to produce the tableau model, in SMV code form. This code will later be combined with the SMV code describing the model the user wants to minimally refine, and fed to the SMV model checker. We chose to implement the tableau construction described in [CGL96], because it is an adequately powerful construct (in the sense that it serves as a tableau for the full language) while being relatively simple (it does not attempt to incorporate complex formalisms for the fairness constraints, but only sets of sets of states). We will review the construction here.

The basic idea is to decompose the given formula ϕ in its sub-formulae and build a model the states of which are appropriate subsets of those (slightly altered) sub-formulae.

Definition 5.3.1 (Sub-formulae and elementary formulae)
The set $\mathrm{sub}(\phi)$ of the sub-formulae of ϕ and the set $\mathrm{el}(\phi)$ of elementary formulae of ϕ are defined as follows.

1. (a) If $\phi = \top$ or $\phi = \bot$, then $\mathrm{sub}(\phi) = \{\phi\}$.
   (b) If $\phi = p$ or $\phi = \neg p$ then $\mathrm{sub}(\phi) = \{\phi, p\}$.
   (c) If $\phi = \mathrm{AX}\psi$, then $\mathrm{sub}(\phi) = \{\phi\} \cup \mathrm{sub}(\psi)$.
   (d) If ϕ is of the form $\phi_1 \vee \phi_2$ or $\phi_1 \wedge \phi_2$ or $\mathrm{A}(\phi_1 \mathrm{U} \phi_2)$ or $\mathrm{A}(\phi_1 \mathrm{R} \phi_2)$, then $\mathrm{sub}(\phi) = \{\phi\} \cup \mathrm{sub}(\phi_1) \cup \mathrm{sub}(\phi_2)$.
2. (a) If $\phi = \top$ or $\phi = \bot$, then $\mathrm{el}(\phi) = \emptyset$.
   (b) If $\phi = p$ or $\phi = \neg p$ then $\mathrm{el}(\phi) = \{p\}$.
   (c) If $\phi = \phi_1 \vee \phi_2$ or $\phi_1 \wedge \phi_2$, then $\mathrm{el}(\phi) = \mathrm{el}(\phi_1) \cup \mathrm{el}(\phi_2)$.
   (d) If $\phi = \mathrm{AX}\psi$, then $\mathrm{el}(\phi) = \{\phi\} \cup \mathrm{el}(\psi)$.
   (e) If $\phi = \mathrm{A}(\phi_1 \mathrm{U} \phi_2)$ or $\phi = \mathrm{A}(\phi_1 \mathrm{R} \phi_2)$, then $\mathrm{el}(\phi) = \{\mathrm{AX}\bot, \mathrm{AX}\phi\} \cup \mathrm{el}(\phi_1) \cup \mathrm{el}(\phi_2)$.

where p is a propositional letter. The formula $\mathrm{AX}\bot$ denotes the lack of fair paths beginning at the state which satisfies this formula. □

The tableau $\tau(\phi)$ of a formula ϕ is the tuple $\langle W, S, \mathcal{A}, v, \rightarrow, \mathcal{F} \rangle$. The state space of $\tau(\phi)$ will be the set $2^{\mathrm{el}(\phi)}$. Before we define the set of initial states and the transition relation, we will define a map sat from $\mathrm{el}(\phi) \cup \mathrm{sub}(\phi) \cup \{\top, \bot\}$ to subsets of $2^{\mathrm{el}(\phi)}$. Intuitively, $\mathrm{sat}(\psi)$ will be the set of states of the tableau that satisfy ψ.

1. $\mathrm{sat}(\psi) = \{s \mid \psi \in s\}$ where $\psi \in \mathrm{el}(\phi)$.
2. $\mathrm{sat}(\neg p) = \{s \mid p \notin s\}$ where p is an atomic proposition.

3. $\mathtt{sat}(\phi_1 \vee \phi_2) = \mathtt{sat}(\phi_1) \cup \mathtt{sat}(\phi_2)$.

4. $\mathtt{sat}(\phi_1 \wedge \phi_2) = \mathtt{sat}(\phi_1) \cap \mathtt{sat}(\phi_2)$.

5. $\mathtt{sat}(\mathrm{A}(\phi_1 \mathrm{U} \phi_2)) = (\mathtt{sat}(\phi_2) \cup (\mathtt{sat}(\phi_1) \cap \mathtt{sat}(\mathrm{AXA}(\phi_1 \mathrm{U} \phi_2)))) \cup \mathtt{sat}(\mathrm{AX}\bot)$.

6. $\mathtt{sat}(\mathrm{A}(\phi_1 \mathrm{R} \phi_2)) = (\mathtt{sat}(\phi_2) \cap (\mathtt{sat}(\phi_1) \cup \mathtt{sat}(\mathrm{AXA}(\phi_1 \mathrm{R} \phi_2)))) \cup \mathtt{sat}(\mathrm{AX}\bot)$.

Definition 5.3.2 (The tableau construction)
The tableau $\tau(\phi) = \langle W, S, \mathcal{A}, v, \rightarrow, \mathcal{F} \rangle$ is defined as follows.

- $W = 2^{\mathrm{el}(\phi)}$.
- $S = \mathtt{sat}(\phi)$.
- $\mathcal{A} = \{p \mid p \in \mathrm{el}(\phi)\}$.
- $v(s) = \{p \mid p \in s\}$.
- $s \rightarrow t$ if and only if $\bigwedge_{\mathrm{AX}\psi \in \mathrm{el}(\phi)} s \in \mathtt{sat}(\mathrm{AX}\psi) \Rightarrow t \in \mathtt{sat}(\psi)$.
- $\mathcal{F} = \{(W \setminus \mathtt{sat}\,(\mathrm{AXA}\,(\phi_1 \mathrm{U} \phi_2))) \cup \mathtt{sat}\,(\phi_2) \mid \mathrm{AXA}\,(\phi_1 \mathrm{U} \phi_2) \in \mathrm{el}\,(\phi)\}$.

The proof that this structure is indeed a tableau of ϕ in the sense of lemma 5.2.5 can be found in [CGL96].

Our implementation consists of a set of classes in C++ that can be used for expressing an arbitrary ACTL formula and generating its tableau model, via definition 5.3.2. Pictures of the model (in the EPS format) can be be produced for visualisation purposes. Moreover, code can be produced that describes the tableau in the SMV format. This implementation can be obtained from `http://www.cs.bham.ac.uk/~nkg/`.

This set of classes comprises of:

- A base class `Formula` and its derived classes, for the representation of ACTL formulae: `Conjunction`, `Disjunction` (for the propositional connectives), `AtomicFormula`, `NegatedAtomicFormula` (for the propositional variables and their negations), `Until` (for AU), `Release` (for AR) and `Next` (for AX). These classes provide auxiliary methods, including methods for extracting the set of elementary formulae and sub-formulae of an instance.

- A class `Model` that handles all issues related to the building, storing, restoring, visualising and converting the model to SMV code.

- A class `Tableau` that is derived from `Model`. This class contains the mechanisms for the building of the tableau model, given a `Formula`-type object that represents an ACTL formula.

The algorithm for producing the tableau is the direct one, i.e. symbolic techniques are have not been used. That limits the usefulness of our implementation because the tableau reviewed above (and all the other tableaus from the literature that have been mentioned previously) is of exponential size in the length of the formula. Therefore, our implementation will have an exponential complexity in the length of the formula, unless significant progress is made in the symbolic techniques for producing these tableaus, or new tableau constructions that do not suffer from the state explosion problem are proposed. We describe possible avenues of research on this issue in section 6.2.

In order to test the implementation and, more importantly, to examine the process from a practical perspective, a case study was performed on the well-known example of *mutual exclusion*. The purpose of this case study is to demonstrate that the need for minimal refinement arises naturally when designing a system, and that minimal refinement can automatically produce models that satisfy the given requirements and are as close to the original model, in terms of refinement.

We chose the mutual exclusion problem because it is a simple, yet non-trivial example of a model/protocol, and because of its ubiquity in textbooks, e.g. in the book [HR00] (pages 181–200). The account appearing in this book is incremental, and the stages shown demonstrate the process of designing a model of a protocol which adheres to an increasing subset of the final specifications.

The problem of mutual exclusion arises in concurrent computation when shared resources are used. In particular, assume that two processes p_1 and p_2 are running on the same computer system by means of some kind of time-sharing.[1] Assume further that, at times, p_1 and p_2 need to use some resource that cannot offer concurrent access, e.g. a printer. It is customary that the system employs some method of mutual exclusion that will ensure that the resource is not accessed simultaneously by the two processes. This can be accomplished through many techniques, either preemptive, such as an operating system-level scheduler, or cooperative, such as the use of semaphores. By *critical section* we will denote the duration within which a process is accessing the resource.

Regardless of the specific techniques used, the scheduling must have a number of properties. The following four are discussed in [HR00]:

Safety: No two processes should ever be in their critical sections at the same time.

Liveness: If a process requests to enter its critical section, it will eventually be allowed to do so.

Non-blocking: A process can always request to enter its critical section.

[1]We will only consider two processes for simplicity.

No strict sequencing: There is no fixed order in which processes must enter their critical section.

We will model the processes from the point of view of the protocol used to enforce mutual exclusion. The processes, then, can be assumed to have three potential states of execution:

- A process can be doing work which is unrelated to the shared resource.
- Or, it may request to enter its critical section.
- Or, it may be executing inside its critical section.

Therefore we will use two propositional variables per process, t_i denoting that process i requests to enter its critical section, and c_i to mean that process i is inside its critical section. Figure 5.2 presents a transition system for a process. We are using an over-simplified model of a process, in the sense that each process has to change state in every clock tick, i.e. it is not allowed to stay in the same state. Also note that the initial state of the process is denoted by a diamond.

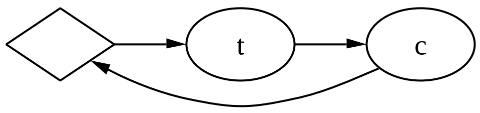

Figure 5.2: A model of a process.

Two of these processes were modelled in SMV code and combined by asynchronous composition in SMV (note that this is not the same as the synchronous product construction). This means that at any given moment one process is chosen non-deterministically and is allowed to execute for one step. The state variables are now named t1, t2, c1, c2. The result is shown in figure 5.3. The solid edges indicate that process 1 is executing and the dashed edges respectively for process 2. It is normally required that the scheduling is fair in that both processes are selected for execution infinitely often. This means that the set of solid edges has to be visited infinitely often by a computation path, and similarly for the set of dashed edges. Notice that the transition model actually constructed by the model checker is more complicated as the asynchronous composition operation is implemented by appropriate insertion of auxiliary variables. The distinction will be suppressed in the following, for reasons of readability of the produced models.

In this example, the constraints laid down earlier can be expressed formally.

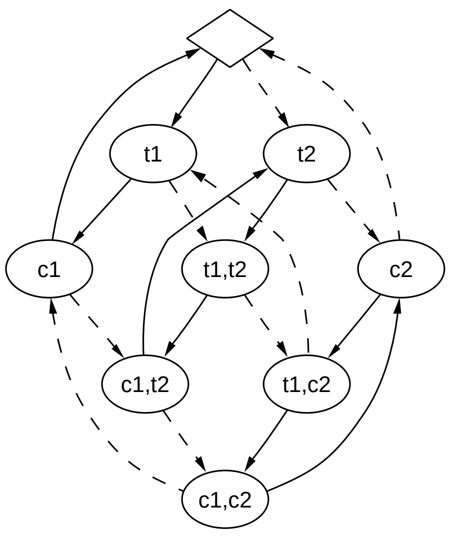

Figure 5.3: M_1, the asynchronous composition of two processes.

Safety: $\mathrm{AG}(\neg c_1 \vee \neg c_2)$.

Liveness: $\mathrm{AG}(t_i \to \mathrm{AF}(c_i))$.

Non-blocking: $\mathrm{AG}(\neg t_i \wedge \neg c_i \to \mathrm{EX}t_i)$.

No strict sequencing: $\mathrm{EG}(c_1 \wedge \mathrm{E}(c_1\mathrm{U}(\neg c_1 \wedge \mathrm{E}(\neg c_2 \mathrm{U} c_1))))$.

Obviously, the non-blocking and non strict sequencing properties cannot be expressed by ACTL formulae, so we will not deal with them. Moreover, they happen to be satisfied by the models we will consider.

A further fairness constraint usually employed in this case is that we do not consider computation paths that spend an inordinate amount of time in a process' critical section. These constraints are expressed by the formulae $\neg c_1$ and $\neg c_2$. In other words, we force the system to only consider paths that visit an infinite number of times the non-critical parts of the processes. We will make use of these constraints.

Obviously, the state $c_1 c_2$ has to be excluded from the set of reachable states. We basically want to impose the formula $\phi = \mathrm{AG}(\neg c_1 \vee \neg c_2)$, an ACTL formula, to the above model. But we want to retain the computational paths that are, in a sense, independent from ϕ. A first test for the minimal refinement operation is to produce $M_1 *_{\mathrm{FTSF}} \phi$.

Concerning the tableau of the safety formula $\phi = \mathrm{AG}(\neg c_1 \vee \neg c_2)$, observe that $\phi = \mathrm{A}(\bot \mathrm{R}(\neg c_1 \vee \neg c_2))$. Therefore, $\mathrm{el}(\phi) = \{\mathrm{AX}\bot, \mathrm{AX}\phi, c_1, c_2\}$, so the tableau can have up to 16 states. However, we may remove the states that are unreachable from the initial ones, as well as the states that have no successors. The resulting model can be seen in figure 5.4. Variables introduced by the tableau construction that have the same truth value across all reachable states have been suppressed. Also, note that there are no fairness constraints.

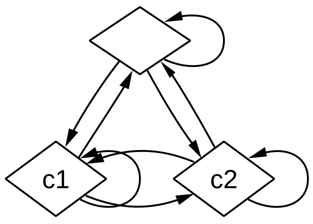

Figure 5.4: The tableau of AG($\neg c_1 \vee \neg c_2$).

Because of the simplicity of the derived tableau, the product structure M_2 of M_1 and τ(AG($\neg c_1 \vee \neg c_2$)) is the expected, i.e. M_1 without the state $c_1 c_2$ (figure 5.5). The fairness constraints remain the same, i.e. the sets of states with $\neg c_1$ and with $\neg c_2$. M_2 is identical the model that appears as the "first attempt" for the mutual exclusion protocol in [HR00], page 182.

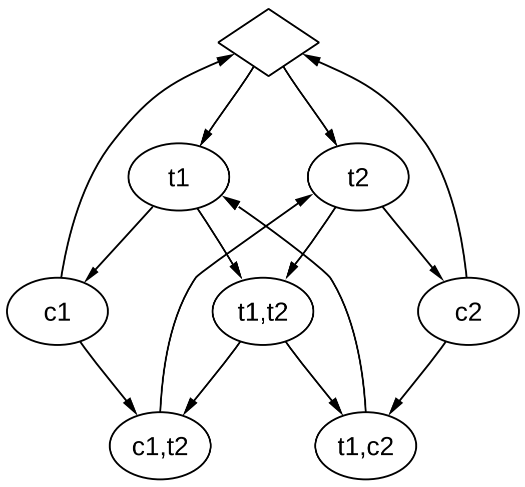

Figure 5.5: M_2, the composition of M_1 and τ(AG($\neg c_1 \vee \neg c_2$)).

We observe that although the safety, non-blocking and no strict sequencing properties are true of M_2, liveness fails. This is because there exists a computation path which goes through the states $t_1 \to t_1, t_2 \to t_1, c_2$ and then loops back to t_1, ad infinitum. This path violates the liveness property because even though process 1 is requesting to enter its critical section, it never manages to do so. We will attempt to repair this by minimally refining M_2 with the liveness property. The liveness property is

$$\text{liveness}_i = \mathrm{AG}(t_i \rightarrow \mathrm{AF}c_i) = \mathrm{AG}(\neg t_i \vee \mathrm{AF}c_i)$$

which is obviously an ACTL formula. The goal is to minimally refine by the formula $\text{liveness}_1 \wedge \text{liveness}_2$. However, this leads to a tableau that is unnecessarily large, so as to increase substantially verification times: notice that since liveness_1 and liveness_2 do not share any propositional variables, the structure $\tau(\text{liveness}_1 \wedge \text{liveness}_2)$ will have exactly as many states as the product of the number of states of the structures $\tau(\text{liveness}_i)$.

Under the present framework this cannot be avoided. However, we can make use of lemma 5.2.7 and perform two minimal refinements. The advantage is that SMV is quite efficient when dealing with product structures so we do not have to re-invent this efficiency for our algorithms. The model for $\tau(\text{liveness}_1)$ is shown in figure 5.6: note that since the model is nearly total, in the sense that every state is connected to almost every other, the edges shown in the figure are the ones that *do not exist* in the actual tableau. Notice that in this tableau, the insertion of auxiliary variables cannot be ignored because one of them $p = \mathrm{AXAF}c_1$ varies in truth value across the reachable states. Moreover, this tableau imposes a non-trivial fairness constraint:

$$(\neg t \wedge \neg p) \vee (t \wedge \neg p \wedge c) \vee (p \wedge c)$$

The states that satisfy this condition are drawn bold in figure 5.6. These must be visited infinitely often by the considered computational paths. The introduction of fairness constraints is due to the need to satisfy eventualities that derive from the *until* operator, implicit in AF.

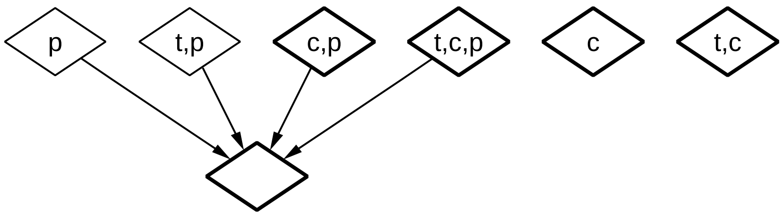

Figure 5.6: $\tau(\text{liveness}_1)$, with edges shown wherever they *do not* exist in the tableau.

Finally, C, the composition of M_2, $\tau(\text{liveness}_1)$ and $\tau(\text{liveness}_2)$ is shown in figure 5.7. Another, perhaps more readable version of C is shown in figure 5.8. Boxes are drawn around states that have identical observable valuations, i.e., in terms of t_i and c_i. Transitions that end at these boxes are meant as abbreviations of a set of transitions leading to each state in the box.

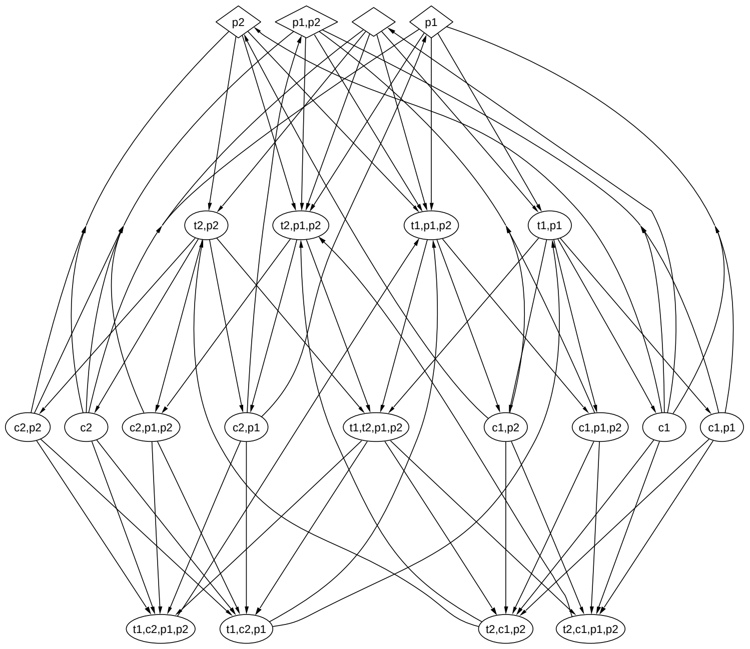

Figure 5.7: C, the composition of M_2, τ(liveness$_1$) and τ(liveness$_2$).

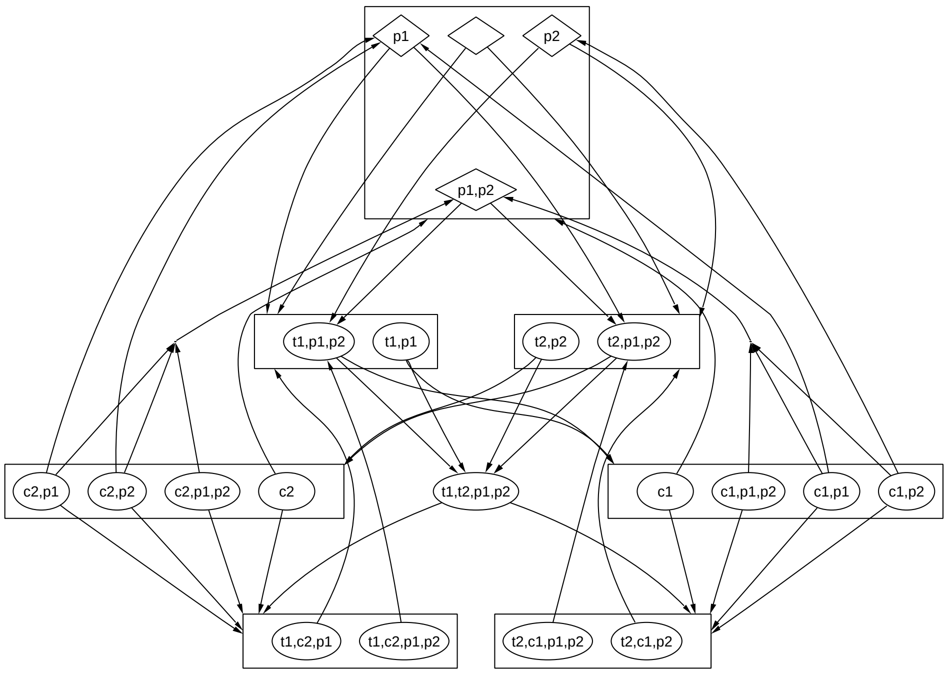

Figure 5.8: C, reformatted for readability.

It is easy to see that the loop $t_1 \to t_1, t_2 \to t_1, c_2$ and back to t_1 still exists, as it should since any finite number of iterations of this loop should be permitted. However, the fairness constraints are now non-trivial because the following ones have been added through the composition with the liveness tableaus:

$$(\neg t_i \wedge \neg p_i) \vee (p_i \wedge c_i)$$

for $i = 1, 2$. This formula is derived from the previously mentioned fairness constraint introduced by $\tau(\text{liveness}_i)$, plus the observation that states that satisfy t_i and c_i do not exist. Given these constraints, it is now impossible to execute the loop $t_1 \to t_1, t_2 \to t_1, c_2$ for an infinite number of times, since such a path would not visit infinitely often (actually, never) a state that satisfies $(\neg t_1 \wedge \neg p_1) \vee (p_1 \wedge c_1)$. Indeed, when model checked by SMV, the model C was found to satisfy the liveness formulae.

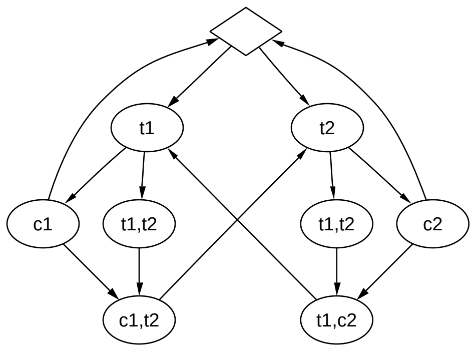

Figure 5.9: C', the "second attempt" for the mutual exclusion protocol.

The model C' appearing in [HR00] as the second attempt for the mutual exclusion protocol is shown in 5.9. As C, it satisfies the safety and liveness properties. Also, it obviously differs from C and is not equivalent to it, neither in terms of fair bisimulation or simulation. However, it can be shown that $C \leftarrow C'$. That is to say, C is strictly closer to the original model M_2 in terms of fair simulation, than C' is. This property, which is to be expected due to the definition of minimal refinement, reflects the fact that in producing C from M_2 no behaviours of M_2 that are compatible with the liveness properties have been removed. On the contrary, C' exhibits a strict temporal behaviour: if process 1 requests to enter its critical section when process 2 is not (state t_1), then it is guaranteed that process 1 will enter its critical section in at most 2 time steps.

This concludes the case study. What has been shown, we believe, is that minimal refinement arises naturally in the process of designing algorithms or models. Moreover, it seems that when it does arise, solutions constructed by hand are sometimes inferior in that they may introduce ad hoc restrictions that do not necessarily follow by the new requirements added to a model. On the other hand, minimal refinement may produce models that are very large in terms of state spaces, and that are difficult to understand.

The SMV code for C that was produced by our implementation, follows. Slight editing by hand has been performed for readability.

```
MODULE processes(c1,t1,c2,t2)
TRANS
        !c1&!t1&!c2&!t2 & !next(c1)&!next(t1)&!next(c2)&next(t2)|
        !c1&!t1&!c2&!t2 & !next(c1)&next(t1)&!next(c2)&!next(t2)|
        !c1&!t1&c2&!t2 & !next(c1)&!next(t1)&!next(c2)&!next(t2)|
        !c1&!t1&c2&!t2 & !next(c1)&next(t1)&next(c2)&!next(t2)|
        !c1&!t1&!c2&t2 & !next(c1)&!next(t1)&next(c2)&!next(t2)|
        !c1&!t1&!c2&t2 & !next(c1)&next(t1)&!next(c2)&next(t2)|
        c1&!t1&!c2&!t2 & !next(c1)&!next(t1)&!next(c2)&!next(t2)|
        c1&!t1&!c2&!t2 & next(c1)&!next(t1)&!next(c2)&next(t2)|
        c1&!t1&c2&!t2 & !next(c1)&!next(t1)&next(c2)&!next(t2)|
        c1&!t1&c2&!t2 & next(c1)&!next(t1)&!next(c2)&!next(t2)|
        c1&!t1&!c2&t2 & !next(c1)&!next(t1)&!next(c2)&next(t2)|
        c1&!t1&!c2&t2 & next(c1)&!next(t1)&next(c2)&!next(t2)|
        !c1&t1&!c2&!t2 & next(c1)&!next(t1)&!next(c2)&!next(t2)|
        !c1&t1&!c2&!t2 & !next(c1)&next(t1)&!next(c2)&next(t2)|
        !c1&t1&c2&!t2 & next(c1)&!next(t1)&next(c2)&!next(t2)|
        !c1&t1&c2&!t2 & !next(c1)&next(t1)&!next(c2)&!next(t2)|
        !c1&t1&!c2&t2 & next(c1)&!next(t1)&!next(c2)&next(t2)|
        !c1&t1&!c2&t2 & !next(c1)&next(t1)&next(c2)&!next(t2)|
        0
INIT
        !c1&!t1&!c2&!t2|
        0
FAIRNESS        !c1
FAIRNESS        !c2

MODULE safety(c1,c2)
TRANS
        !c1&!c2 & !next(c1)&!next(c2)|
        !c1&!c2 & next(c1)&!next(c2)|
        !c1&!c2 & !next(c1)&next(c2)|
        c1&!c2 & !next(c1)&!next(c2)|
        c1&!c2 & next(c1)&!next(c2)|
        c1&!c2 & !next(c1)&next(c2)|
        !c1&c2 & !next(c1)&!next(c2)|
        !c1&c2 & next(c1)&!next(c2)|
        !c1&c2 & !next(c1)&next(c2)|
        0
INIT
        !c1&!c2|
        c1&!c2|
        !c1&c2|
        0

MODULE liveness(t,c,p)
```

```
TRANS
        !t&!p&!c & !next(t)&!next(p)&!next(c)|
        !t&!p&!c & !next(t)&next(p)&!next(c)|
        !t&!p&!c & next(t)&next(p)&!next(c)|
        !t&!p&!c & !next(t)&!next(p)&next(c)|
        !t&!p&!c & next(t)&!next(p)&next(c)|
        !t&!p&!c & !next(t)&next(p)&next(c)|
        !t&!p&!c & next(t)&next(p)&next(c)|
        !t&p&!c & !next(t)&next(p)&!next(c)|
        !t&p&!c & next(t)&next(p)&!next(c)|
        !t&p&!c & !next(t)&!next(p)&next(c)|
        !t&p&!c & next(t)&!next(p)&next(c)|
        !t&p&!c & !next(t)&next(p)&next(c)|
        !t&p&!c & next(t)&next(p)&next(c)|
        t&p&!c & !next(t)&next(p)&!next(c)|
        t&p&!c & next(t)&next(p)&!next(c)|
        t&p&!c & !next(t)&!next(p)&next(c)|
        t&p&!c & next(t)&!next(p)&next(c)|
        t&p&!c & !next(t)&next(p)&next(c)|
        t&p&!c & next(t)&next(p)&next(c)|
        !t&!p&c & !next(t)&!next(p)&!next(c)|
        !t&!p&c & !next(t)&next(p)&!next(c)|
        !t&!p&c & next(t)&next(p)&!next(c)|
        !t&!p&c & !next(t)&!next(p)&next(c)|
        !t&!p&c & next(t)&!next(p)&next(c)|
        !t&!p&c & !next(t)&next(p)&next(c)|
        !t&!p&c & next(t)&next(p)&next(c)|
        t&!p&c & !next(t)&!next(p)&!next(c)|
        t&!p&c & !next(t)&next(p)&!next(c)|
        t&!p&c & next(t)&next(p)&!next(c)|
        t&!p&c & !next(t)&!next(p)&next(c)|
        t&!p&c & next(t)&!next(p)&next(c)|
        t&!p&c & !next(t)&next(p)&next(c)|
        t&!p&c & next(t)&next(p)&next(c)|
        !t&p&c & !next(t)&next(p)&!next(c)|
        !t&p&c & next(t)&next(p)&!next(c)|
        !t&p&c & !next(t)&!next(p)&next(c)|
        !t&p&c & next(t)&!next(p)&next(c)|
        !t&p&c & !next(t)&next(p)&next(c)|
        !t&p&c & next(t)&next(p)&next(c)|
        t&p&c & !next(t)&next(p)&!next(c)|
        t&p&c & next(t)&next(p)&!next(c)|
        t&p&c & !next(t)&!next(p)&next(c)|
        t&p&c & next(t)&!next(p)&next(c)|
        t&p&c & !next(t)&next(p)&next(c)|
        t&p&c & next(t)&next(p)&next(c)|
        0
INIT
        !t&!p&!c|
        !t&p&!c|
        t&p&!c|
        !t&!p&c|
        t&!p&c|
        !t&p&c|
        t&p&c|
        0
FAIRNESS
        !t &!p &!c|
        !t &!p & c|
         t &!p & c|
        !t & p & c|
         t & p & c|
        0


MODULE main
VAR
        t1 : boolean;
        t2 : boolean;
        c1 : boolean;
        c2 : boolean;
        p1 : boolean;
        p2 : boolean;
        procs     : processes(c1,t1,c2,t2);
        safety    : safety(c1,c2);
        liveness1 : liveness(t1,c1,p1);
        liveness2 : liveness(t2,c2,p2);
```

Chapter 6

Conclusions and Further Work

The conclusions, unresolved problems and open questions encountered while researching the topics of this thesis are presented in this chapter.

6.1 Implementations for theory change and fault diagnosis

In chapter 3, Binary Decision Diagrams were used as a basis for the construction of algorithms that implement several types of propositional theory change. The aim was to demonstrate a framework that is general enough, thereby enabling the implementation of many proposed forms of theory change, while at the same time exploiting the benefits in efficiency that a technology like BDDs has to offer. Generality was achieved firstly by providing results that operate on the premise that the theory revision or update operation is defined by a faithful assignment (sections 3.2.1 and 3.2.2). Moreover, several proposals from the literature of theory change were implemented (sections 3.2.3, 3.2.4, 3.2.5 and 3.2.6), demonstrating that even if the operation is not defined through a faithful assignment, in many cases it is possible to construct a BDD algorithm for it.

In order to evaluate these algorithms, upper bounds of the sizes of the main BDDs involved were produced by providing boolean circuit implementations that characterise them through theorem 2.2.1. In all of the cases considered, the bounds produced were very satisfactory, i.e., linear or polynomial. Attempts were also made to estimate the time complexity of these algorithms, by producing upper bounds of it, using the known upper bounds for the primitive operations on BDDs. These attempts were not enlightening, due to the fact that these known upper bounds for the primitive operations are too pessimistic (cf. the bound for the `andExists` algorithm) and, therefore, of little use in estimating the complexity of compound operations.

Possible avenues of further research on the issue of these complexities are

- Maybe it would be profitable to examine a BDD algorithm as a whole, i.e., as an operation that is not understood in terms of the BDD primitives it uses. This attempt would be a major undertaking in terms of work, though.
- A more careful examination of the interactions of the BDD primitives used, maybe in the context of assumptions on the nature of formulae given to the theory change operation, would be meaningful, especially in conjunction with further research on the complexities of the primitive operations.
- Lastly, *lower* bounds for these complexities, in the form of hardness or completeness results, may be obtainable by combining results on the theoretical complexity of the theory change operations [EG92, Neb96, LS96, CDLS99], with results concerning the behaviour of decision problems when inputs are offered in BDD form [FKVV99].

Orthogonal to these possibilities is the question of how to implement *query answering*, rather than computing the representation of the changed epistemic state. In some cases, the use of theory change does not aim to compute the epistemic state per se, but in using it in finding out whether a certain property is entailed. For example, if ϕ is to be changed by an operation $*$ under the formula ψ, we may be interested only in finding out whether $\phi * \psi \models \chi$ for some formula χ. In this case, it may be more profitable in terms of efficiency to provide a single algorithm for this operation, rather than two separate ones for $*$ and $\models$. For propositional logic, query answering is equivalent to $\mathrm{mod}(\phi * \psi) \subseteq \mathrm{mod}(\chi)$ or $\mathrm{mod}(\phi * \psi) \cap \mathrm{mod}(\neg\chi) = \emptyset$. Such an emptiness check may be easier to perform on-the-fly rather than by first computing $\mathrm{mod}(\phi * \psi)$ and then checking for inclusion.

In section 3.3.1, a formulation for fault diagnosis of combinational circuits was presented and benchmarked. The obtained results seem very promising, showing a quadratic complexity for the fault diagnosis of a n-bit adder. However, these results cannot be regarded as conclusive, and further research is possible in trying to get a better picture of the complexities involved. Firstly, the input of the diagnosis algorithm was a set of uniformly distributed random bit-vectors. In reality, the faults usually observed do not produce a uniform distribution of probability on the observations. Therefore, we could re-run the experiments described in section 3.3.3, with appropriately selected input observations. More importantly, it is a fact that a n-bit adder is not a 'hard' circuit to diagnose. In this respect, the experiments could be modified in order to benchmark diagnosis for other circuits. In particular, good candidates would be circuits selected from the ISCAS'85 collection for fault diagnosis.

What has been demonstrated, I believe, is the practicality of the approach contrary to the order of magnitude of the upper bounds calculated; for fault diagnosis, with a moderately-sized computer system one can perform single runs of the diagnosis algorithm on reasonably large circuits. In particular, it has been successfully executed for several times on adders of 1000 bits with each run taking 20 minutes on average. Note that in the model presented, a 1000-bit adder leads to a state-space of 2^{12000} interpretations.

6.2 Minimal refinement

Minimal refinement, described in sections 1.2, 1.3 and in chapters 4 and 5, is a method for changing a given model so that the result refines it minimally while satisfying a new requirement.

Using minimal refinement, a designer can obtain a revised design out of an old one and a new requirement. Moreover, the relationship between the resulting model and the initial one is well-defined: all the behaviours the newly obtained one exhibits are behaviours allowed by the initial system, and, moreover, the set of behaviours is maximal in that it is the largest one that satisfies the new property.

Minimal refinement has been studied under three frameworks:

Section 4.3: In modal logic, we studied minimal refinement over the class of m-saturated models with sets of sentences as the requirements (local and global satisfaction).

Section 4.4: Again, in modal logic, minimal refinement over finite structures was studied, with modal formulae as requirements (local and global satisfaction).

Chapter 5: Finally, we studied minimal refinement over transition systems with fairness constraints with formulae of ACTL as requirements.

In all of these cases, the conditions under which the minimal refinement is non-trivial were characterised (lemmas 4.3.6, 4.4.4 and 5.2.1). In the temporal case as well as in the finite modal one, non-triviality was proved to be decidable (in the same lemmas). The ordering $\leq_M$ was proved to be stoppered (theorems 4.3.10, 4.4.5 and 5.2.6). Lastly, effective ways to represent minimal refinements in the finite modal case and the temporal one, were presented in results 4.4.6, 4.4.7, 4.4.8 and 5.2.6.

Finally, a sample implementation of minimal refinement in the case of ACTL was presented in section 5.3, along with a detailed study of an example of a mutual exclusion protocol. Using this implementation, a designer can provide an ACTL formula ϕ and a system M described in the language of the SMV model checker. The implementation will then produce the minimal refinement $M *_{\mathrm{FTSF}} \phi$, coded in the SMV language. The result can then be fed into the SMV model checker where the designer can verify the result against any further requirements.

Many possibilities exist for further work, as many questions are open with respect to minimal refinement. For the case of modal logic and finite structures, an obvious research goal is to find tractable algorithms for constructing minimal models. For the moment, the obvious and naive algorithms for the modal case of minimal refinement are of non-deterministic exponential complexity.

Studying the computational complexity of the associated problems is another obvious research avenue. It is easy to see that these problems will have an expensive worst-case complexity. After all, modal satisfiability is PSPACE-complete, regardless of bounds on the number of propositional atoms or modalities in the language [Hal95], and the test for non-triviality can easily be modified so as to solve modal satisfiability, rendering it PSPACE-hard in the length of the formula. On the other hand, it is also easy to show that its complexity is linear in the size of the model provided.

In conjunction to this, additional work can be focused on characterising more precisely the structure of the set of the resulting models, since we have obtained only very pessimistic upper bounds on its size.

In the case of temporal logics, of course it would be desirable to extend the set of results to languages that are not universally quantified (like ACTL) and therefore are not preserved by refinement. Unfortunately, as the results we have obtained on ACTL depend crucially on this property, they are not transferable to such new languages and therefore a whole new approach would need to be developed.

Another direction for future work is the investigation of the tractability of the method in the case of ACTL. We are currently using the tableau construction introduced in [CGL96], which is of exponential size in the length of the formula. Thus, the minimal refinement may be of intractable size, as it is the product of the initial model and the tableau. One improvement that comes at the cost of some expressiveness is to use safety-ACTL[1], for which there is a more compact tableau [KGG99], although still exponential in the worst case.

Further research could focus on ways to address this state-explosion problem from its implementation aspect, either by considering improvements on the tableau construction or by looking into the application of symbolic methods for generation of $M *_{\mathrm{FTSF}} \phi$.

[1]Safety-ACTL is the fragment of ACTL obtained by only using AX and AW (the weak counterpart of *until*).

Bibliography

[AGM85] C. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: partial meet contraction and revision functions. *Journal of Symbolic Logic*, 50:510–530, 1985.

[AH97] H. R. Andersen and H. Hulgaard. Boolean expression diagrams. In *Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 88–98, Warsaw, Poland, 1997. IEEE Computer Society.

[AL91] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.

[And97] H. R. Andersen. An introduction to binary decision diagrams, October 1997. Lecture Notes, URL: `http://www.it.dtu.dk/~hra`.

[BB95] C. Boutilier and V. Becher. Abduction as belief revision. *Artificial Intelligence*, 77(1):43–94, 1995.

[BCM$^+$90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.

[BdRV01] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.

[BFG$^+$91] A. Bouajjani, J. C. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis. Safety for branching time semantics. In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming, ICALP'91*, volume 510 of *LNCS*, pages 76–92, Madrid, Spain, July 1991. Springer.

[BMP97] H. Bezzazi, D. Makinson, and R. P. Pérez. Beyond rational monotony: Some strong non-horn rules for nonmonotonic inference relations. *Journal of Logic and Computation*, 7(5):605–631, 1997.

[Bor85] A. Borgida. Language features for flexible handling of exceptions in information systems. *ACM Transactions on Database Systems (TODS)*, 10(4):565–603, 1985.

[Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[Bry91] R. E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2):205–213, February 1991.

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[Bry95] R. E. Bryant. Binary decision diagrams and beyond: Enabling technologies for formal verification. In *International Conference on Computer-Aided Design ICCAD'95*, pages 236–243, November 1995.

[CDLS99] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. The size of a revised knowledge base. *Artificial Intelligence*, 115(1):25–64, 1999.

[CE81] E. M. Clarke and E. A. Emerson. The design and synthesis of synchronization skeletons using temporal logic. In *Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–72, Yorktown Heights, New York, 1981. Springer-Verlag.

[CES86] E. M. Clarke, E.A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CGL96] E. Clarke, O. Grumberg, and D. Long. Model checking. Nato ASI Series F, volume 152, Springer-Verlag, 1996. Marktoberdorf summer school.

[Che80] B.F. Chellas. *Modal Logic: an introduction.* Cambridge University Press, 1980.

[Dal88] M. Dalal. Investigations into a theory of knowledge base revision: Preliminary report. In NCAI'88 [NCA88], pages 475–479.

[dR95] M. de Rijke. Modal model theory. Technical Report CS–R9517, CWI, Amsterdam, 1995.

[dV94] A. del Val. On the relation between the coherence and foundations theories of belief revision. In *Proceedings of the Twelfth American National Conference on Artificial Intelligence*, pages 909–914, 1994.

[EG92] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *Artificial Intelligence*, 52(2–3):227–270, 1992.

[FH99] N. Friedman and J. Y. Halpern. Belief revision: A critique. *Journal of Logic, Language, and Information*, 8:401–420, 1999.

[FKVV99] J. Feigenbaum, S. Kannan, M. Y. Vardi, and M. Viswanathan. Complexity of problems on graphs represented as OBDDs. *Chicago Journal of Theoretical Computer Science*, 1999(5), August 1999.

[For89] K. D. Forbus. Introducing actions into qualitative simulation. In IJCAI89 [IJC89], pages 1273–1278.

[Gär88] P. Gärdenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. Bradford Books, MIT Press, Cambridge, Mass, 1988.

[Gär92a] P. Gärdenfors, editor. *Belief Revision*. Cambridge Computer Tracts. Cambridge University Press, Cambridge, 1992.

[Gär92b] P. Gärdenfors. Belief revision: An introduction. In *Belief Revision* [Gär92a], pages 1–20.

[GL94] O. Grumberg and D. E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems*, 16(3):843–871, May 1994.

[GM88] P. Gärdenfors and D. Makinson. Revisions of knowledge systems using epistemic entrenchment. In M. Vardi, editor, *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 83–95, Los Altos, CA, 1988. Morgan Kaufmann.

[GM95] C. Grahne and A. O. Mendelzon. Updates and subjunctive queries. *Information and Computation*, 116:241–252, 1995.

[GPS98] C. Gröpl, H. J. Prömel, and A. Srivastav. Size and structure of random ordered binary decision diagrams (extended abstract). In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS'98)*, volume 1373 of *Lecture Notes in Computer Science*, pages 238–248. Springer, 1998.

[GR02a] N. Gorogiannis and M. D. Ryan. Implementation of belief change operators using binary decision diagrams. *Studia Logica*, 70:131–156, 2002.

[GR02b] N. Gorogiannis and M. D. Ryan. Requirements, specifications and minimal refinement. In *9th Workshop on Logic, Language, Information and Computation*, volume 67 of *Electronic Notes in Theoretical Computer Science*, Brazil, September 2002. Elsevier.

[Gra98] C. Grahne. Updates and counterfactuals. *Journal of Logic and Computation*, 8:87–117, 1998.

[Gro88] A. Grove. Two modelings for theory change. *Journal of Philosophical Logic*, 17:157–170, 1988.

[GS88] M. L. Ginsberg and D. E. Smith. Reasoning about action i: a possible worlds approach. *Artificial Intelligence*, 35:165–195, 1988.

[GZ01] J. F. Groote and H. Zantema. Resolution and binary decision diagrams cannot simulate each other polynomially (extended abstract). In *Proceedings of the 4th International Andrei Ershov Memorial Conference, Perspectives of System Informatics, (PSI'01)*, volume 2244 of *Lecture Notes in Computer Science*, pages 33–38. Springer, 2001.

[Hal95] J. Y. Halpern. The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic. *Artificial Intelligence*, 75(2):361–372, 1995.

[HKR97] T. A. Henzinger, O. Kupferman, and S. K. Rajamani. Fair simulation. In *Proceedings of the Ninth International Conference on Concurrency Theory (CONCUR)*, volume 1243 of *Lecture Notes in Computer Science*, pages 273–287. Springer-Verlag, 1997.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, January 1985.

[Hol95] M. Hollenberg. Hennessy-Milner classes and process algebra. In A. Ponse, M. de Rijke, and Y. Venema, editors, *Modal Logic and Process Algebra*, pages 187–216. CSLI Publications, 1995.

[HR00] M. R. Huth and M. D. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2000.

[HR02] H. Harris and M. Ryan. Feature integration as an operation of theory change. In F. van Harmelen, editor, *Proceedings of ECAI 2002, 15th European Conference on Artificial Intelligence*, pages 546–550. IOS Press, 2002.

[IJC89] *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, USA, August 1989.

[KGG99] S. Katz, O. Grumberg, and D. Geist. "Have I written enough properties?" - A method of comparison between specification and implementation. In *Conference on Correct Hardware Design and Verification Methods*, pages 280–297, 1999.

[KLM90] S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44:167–207, 1990.

[KM89] H. Katsuno and A. O. Mendelzon. A unified view of propositional knowledge base updates. In IJCAI89 [IJC89], pages 1413–1419.

[KM91] H. Katsuno and A. O. Mendelzon. Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52:263–294, 1991.

[KM92] H. Katsuno and A. O. Mendelzon. On the difference between updating a knowledge base and revising it. In Gärdenfors [Gär92a], pages 183–203.

[KP00] S. Konieczny and R. P. Pérez. A framework for iterated revision. *Journal of Applied Non-Classical Logics*, 10(3–4):339–367, December 2000.

[LN] J. Lind-Nielsen. BuDDy — a binary decision diagram package. URL: `http://www.itu.dk/research/buddy/`.

[LS95] P. Liberatore and M. Schaerf. Relating belief revision and circumscription. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1557–1563, Los Altos, 1995. Morgan Kaufmann.

[LS96] P. Liberatore and M. Schaerf. The complexity of model checking for belief revision and update. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 556–561. AAAI Press/The MIT Press, 1996.

[Mai00] M. Maidl. The common fragment of CTL and LTL. In *Proceedings of the 41th Annual Symposium on Foundations of Computer Science*, pages 643–652, 2000.

[MC91] J. C. Madre and O. Coudert. A logically complete reasoning maintenance system based on a logical constraint solver. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 294–299, Sydney, Australia, August 1991. Morgan Kaufmann.

[McM93] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, pages 481–489, London, UK, September 1971.

[Nay94] A. C. Nayak. Iterated belief change based on epistemic entrenchment. *Erkenntnis*, 41:353–390, 1994.

[NCA88] *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, Minnesota, August 1988.

[Neb96] B. Nebel. How hard is it to revise a belief base? Technical Report 83, Institut für Informatik, Albert-Ludwigs-Universität Freiburg, August 1996.

[PMT02] H. Peng, Y. Mokhtari, and S. Tahar. Environment synthesis for compositional model checking. In *Proceedings of the IEEE International Conference on Computer Design*, pages 70–75. IEEE Computer Society Press, September 2002.

[PR99] M. Plath and M. D. Ryan. SFI: a feature integration tool. *Advances in Computing Science*, Tool Support for System Specification, Development and Verification:201–216, 1999.

[PR01] M. Plath and M. D. Ryan. Feature integration using a feature construct. *Science of Computer Programming*, 41(1):53–84, 2001.

[Rei87] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[Rot99] H. Rott. Coherence and conservatism in the dynamics of belief. part i: Finding the right framework. *Erkenntnis*, 50:387–412, 1999.

[RS97] M. D. Ryan and P. Schobbens. Counterfactuals and updates as inverse modalities. *Journal of Logic, Language and Information*, 6(2):123–146, 1997.

[Sat88] K. Satoh. Nonmonotonic reasoning by minimal belief revision. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 455–462, Tokyo, Japan, 1988.

[Som] F. Somenzi. CUDD: CU decision diagram package. URL: `http://vlsi.colorado.edu/~fabio/CUDD/`.

[Som99] F. Somenzi. Binary decision diagrams. In M. Broy and R. Steinbruggen, editors, *Calculational System Design*, volume 173 of *NATO Science Series F: Computer and Systems Sciences*, pages 303–366. IOS Press, 1999.

[vBvEF93] J. van Benthem, J. van Eijck, and A. Frolova. Changing preferences. Technical Report CS-93-10, Centre for Mathematics and Computer Science, Amsterdam, 1993.

[vdM91] R. van der Meyden. A clausal logic for deontic action specification. In *Proceedings of the International Logic Programming Symposium*, pages 221–238, San Diego, October 1991. MIT Press.

[Wil97] M. Williams. Anytime belief revision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 74–80. Morgan Kaufmann, 1997.

[Win88] M. Winslett. Reasoning about action using a possible models approach. In NCAI'88 [NCA88], pages 89–93.

[WJP01] É. Würbel, R. Jeansoulin, and O. Papini. Spatial information revision: a comparison between 3 approaches. In *Sixth European Conference on Symbolic and Quantitatives Approaches to Reasoning with Uncertainty*, volume 2143 of *Lecture Notes in Artificial Intelligence*, pages 454–465, 2001.

[ZF96] Y. Zhang and N. Y. Foo. Updating knowledge bases with disjunctive information. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference*, volume 1, pages 562–568, Portland, Oregon, August 1996. AAAI Press/The MIT Press.