Multi-Valued Abstraction and Compositional Model Checking

Yael Meller

Multi-Valued Abstraction and Compositional Model Checking

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Yael Meller

Submitted to the Senate of
the Technion — Israel Institute of Technology
Chislev 5770 Haifa December 2009

The research thesis was done under the supervision of Prof. Orna Grumberg in the Computer Science Department.

I would like to thank Orna for being the best supervisor I could have hoped for and so much more. Thank you for introducing me to the wonderful world of research, and for the constant guidance and support throughout the way. More importantly, thank you for the friendship, which made my studies so enjoyable.
I would also like to thank my family, my husband Nimrod, his parents Daniella and Isaac and my parents Tami and Kobi, for the constant encouragement and help in juggling between studies and raising the girls.

The generous financial support of the Technion is gratefully acknowledged.

Contents

List of Figures

Abstract

We present a framework for fully automated compositional verification of μ-calculus specifications over multi-valued systems, based on multi-valued abstraction and refinement.

Multi-valued models are widely used in many applications of model checking. They enable a more precise modeling of systems by distinguishing several levels of uncertainty and inconsistency. Successful verification tools such as STE (for hardware) and YASM (for software) are based on multi-valued models.

Our compositional approach model checks individual components of a system. Only if all individual checks return *indefinite* values, the *parts of the components* which are responsible for these values, are composed and checked. Thus the construction of the full system is avoided. If the latter check is still indefinite then a *refinement* is needed.

We formalize our framework based on bilattices, consisting of a truth lattice and an information lattice. Formulas interpreted over a multi-valued model, are evaluated with respect to the truth lattice. On the other hand, refinement is now aimed at increasing the information level of model details, thus also increasing the information level of the model checking result. Based on the two lattices and on multi-valued model checking graphs, we suggest how multi-valued models should be composed, checked, and refined.

Abbreviations and Notations

$\mathcal{D}$	—	De Morgan lattice
AP	—	Set of atomic propositions
L_μ	—	The μ-calculus
M	—	Multi-Valued Kripke model
φ	—	μ-calculus formula
$Sub(\varphi)$	—	Subformulas of φ
$fp(Z)$	—	Fixpoint formula associated with variable Z
$\mathcal{V}$	—	Environment, explaining the meaning of free variables in the formula
$\|\varphi\|_{\mathcal{V}}^{M}$	—	Semantics of φ w.r.t. a model M and an environment $\mathcal{V}$
$G(M, \varphi)$	—	mc-graph on a model M and a formula φ
χ	—	mc-function
$\leq_i$	—	Information order
$\leq_t$	—	Truth order
$true$	—	Truth value ‘true’, the top of the truth order
$false$	—	Truth value ‘false’, the bottom of the truth order
$\top$	—	Truth value ‘top’, the top of the information order
$\bot$	—	Truth value ‘bottom’, the bottom of the information order
$\wedge$	—	Meet of truth lattice
$\vee$	—	Join of truth lattice
$\otimes$	—	Meet of information lattice
$\oplus$	—	Join of information lattice
$\mathcal{B}(\mathcal{D})$	—	The bilattice induced by De Morgan algebra $\mathcal{D}$
CPDB	—	Consistent Partial Distributive Bilattice
$\mathcal{P}(\mathcal{B}(\mathcal{D}))$	—	The CPDB induced by bilattice $\mathcal{B}(\mathcal{D})$
$\preceq$	—	Mixed simulation relation
$M_1 \| M_2$	—	Composition of models M_1 and M_2

Chapter 1

Introduction

In this work we present a framework for fully automated compositional verification of μ-calculus specifications over multi-valued systems, based on multi-valued abstraction and refinement. Our interest in such a framework stems from the fact that multi-valued modeling is widely used in many applications of model checking. It is used both to model concrete systems more precisely and to define abstract models.

Multi-valued models enable a more precise modeling of systems by distinguishing between several levels of uncertainty and inconsistency [6, 10, 7, 24, 40, 37]. These models have been widely used for abstraction as well [50, 51, 3, 31, 36, 32]. Tools to provide multi-valued verification such as YASM ([36]) and STE ([50]) were developed and successfully applied to software and hardware verification.

Multi-valued models, both abstract and concrete, may still suffer from the state explosion problem. Thus, a compositional approach may enhance the verification of larger systems.

The first step we take in formalizing a compositional multi-valued framework is to consider bilattices [26] as part of our framework. A bilattice defines two lattices over a given set of elements: the *truth lattice* and the *information lattice*, each accompanied with an order. Formulas interpreted over a multi-valued model are evaluated with respect to the truth lattice. On the other hand, the relation of "more abstract" over models is based on the information lattice: Roughly, a model M_2 is more abstract than a model M_1 if values of atomic propositions and transitions in M_2 are smaller or equal by the information order than the corresponding values in M_1. Consequently, the valuation of a formula in M_2 will be smaller or equal by the information order than its value in M_1. In fact, since we consider the full μ-calculus, a bidirectional correspondence between transitions of M_1 and M_2 is needed. To capture this, we define a mixed-simulation relation, based on the information lattice.

Bilattices provide a natural way to identify lattice elements that are *consistent*, meaning that they represent some concrete elements of the bilattice (to be formalized later). We can also identify elements that are *definite*. Those are the elements that need not be refined anymore. In most of the work we restrict the discussion to Consistent Partial Distributive Bilattices (CPDB), which consist of exactly all the consistent elements. In Chapter 7 we consider also full distributive bilattices. In particular, we discuss the interesting special case of the 4-valued Belnap bilattice.

We attempt to address compositional verification for our context in a similar manner to [54]. There, abstraction and compositional verification are joined in the context of 3-valued abstraction: each component M_i of a composed system M is lifted into a 3-valued model $M_i\uparrow$ which forms an abstraction of M. Model checking a formula φ on $M_i\uparrow$ can result in either a definite value *true* or *false*, or an *indefinite* value. In the former case, it is guaranteed that the result is also the value of φ on M. In the latter case, however, nothing can be deduced about the composed system. If the checks of all individual components return *indefinite* values, then the *parts of the components* which are responsible for these values are identified, composed, and model checked. Thus, the construction of the fully composed system is avoided. Finally, if the check of the partially composed system is still indefinite then a *refinement* is applied to each component separately.

For our multi-valued framework, once we establish our setting by means of bilattices, we can fill in the rest of the framework's ingredients. First, we define the notion of *composition* of multi-valued systems. Next, for model checking we use the model checking algorithm for multi-valued systems and the alternation-free μ-calculus, suggested in [53]. We also show, in case the checks on individual components are indefinite, how to identify, compose, and check the parts of the models that are needed for the checked formula. As we exemplify later, the resulting composed system is often much smaller than the full composed system. Finally, we develop a heuristic for finding a *criterion for refinement*, in case the model checking of the composed system returns an indefinite result.

In the framework above we do not discuss the construction of multi-valued abstract models. This is investigated for instance in [38], which presents a methodology for a systematic construction of an abstract model for a given concrete one.

Other works deal with several aspects of multi-valued model checking, but none investigate a compositional approach. Multi-valued symbolic model checking is described in [11]. An alternative definition of (bi)simulation is suggested in [43]. However, there, the relation returns a value, indicating how "close" the models are. Our mixed simulation, on the other hand, returns either true or false, indicating whether one model is an abstraction of the other. A relation similar to our mixed simulation is defined in [3]. Preservation of formulas via simulation is described there in terms of an information order. However, they do not handle a general multi-valued framework, but rather a 6-valued one. Also, they suggest refinement only if the result is the smallest element in the information order, $\perp$. In contrast, we allow refinement for any indefinite value in the bilattice. Bilattices are used also in [38]. However, they are not exploited there for refinement.

To summarize, the main contributions of this work are:

- We present a framework for fully automated compositional verification of multi-valued systems with respect to μ-calculus specifications. The framework is based on multi-valued abstraction-refinement. To the best of our knowledge, this is the first compositional approach for multi-valued model checking.
- We apply our framework to the alternation-free μ-calculus model checking algorithm. In particular, we develop an algorithm for refinement in this context.
- We formalize our framework based on bilattices, consisting of a truth lattice and an information lattice. This allows to naturally define the consistent and definite elements in the bilattice. It also provides a clear definition of abstraction and refinement in the multi-valued context. It thus provides a better understanding of the multi-valued framework.
- Based on the information order of a bilattice, we define a mixed simulation relation over multi-valued models, preserving μ-calculus specifications.

1.1 Related Work

Multi-Valued Model Checking

Multi-valued model checking is a generalization of classical model-checking [14, 12, 8]. In multi-valued model checking both atomic propositions and transitions in the model and the satisfaction relation can be multi-valued.

Multi-valued model checking has many important applications within the verification framework. For example, 3-valued models are used to describe models with partial information [6]. 4-valued models can model disagreement and their generalizations are used to handle inconsistent views of a system [24, 40]. Temporal logic query checking [10, 7, 37] can also be reduced to multi-valued model checking. In probabilistic modeling, transitions are labelled with values representing the probability that they are taken. The satisfaction relation then becomes the probability that a desired property holds [23].

Multi-valued models have been widely used for abstraction as well: 3-valued (abstract) models allow proving truth as well as falsity of formulas for the concrete models they represent [51, 31, 32]. The 6-valued models in [3] are tuned to better achieving proofs of falsification. 4-valued models extend 3-valued abstractions by enabling to capture inconsistencies in software [36] and hardware (in STE) [50]. Tools to provide multi-valued verification such as YASM ([36]) and STE ([50]) were developed and successfully applied to software and hardware verification.

There are several ways of handling the multi-valued model checking problem. One way is the reduction approach, where the problem is reduced to several traditional 2-valued or 3-valued problems [29, 42, 35, 8]. As opposed to the reduction approach, the direct approach checks the property directly on the multi-valued structure [15, 13, 53].

In our work we present an algorithm for abstraction-refinement of multi-valued models, which is based on direct model checking of the property, specifically, on the algorithm presented in [53].

The domain of logical values used in multi-valued model-checking is given by a finite distributive lattice. This lattice can be referred to as “truth lattice”. There are several works where the logical values are presented also with an information ordering, which is either a lattice or a partial lattice. This is captured by the notion of bilattices [26]. [38] uses the notion of bilattices and information order when presenting a framework for creation of precise abstractions. In [3] a 6-valued logic is presented as a bilattice, and the information order is used to define mixed simulation between models. The 4-valued Belnap logic [4], with both information and truth orders, is used in software modelling ([36]). There, the 4 values of the logic are used to distinguish between non-determinism and lack of information. In STE the 4-valued Belnap logic is used for hardware modelling ([50]). The 4 values are used in STE to distinguish between inconsistency which results from contradiction of the requirements and the behavior of the circuit, and lack of information.

We also use bilattices to present multi-valued logics. We define a mixed simulation relation between models based on bilattices (Section 4.1). The mixed simulation for the 6-valued logic described in [3] is a particular case of our mixed simulation. Defining mixed simulation sets the basis for describing a framework for abstraction-refinement of multi-valued models (Chapter 4), and a compositional framework (Chapter 6).

Abstraction-Refinement

Abstractions [18] hide certain details of the system in order to result in a smaller (abstract) model. They are designed to be *conservative*. That is, from checking properties on the abstract model, their truth value on the concrete model can be concluded. Various abstraction techniques are formalized in the framework of abstract interpretation [21, 22]. Commonly used abstractions are conservative with respect to validity of properties, yielding false-negative results. In contrast, *3-valued abstractions* [6, 39, 30] are conservative with respect to both validity and falsity of properties. They never yield false-negative or false-positive results. However, they may produce an *indefinite* result.

Multi-valued semantics enable a more precise modeling of systems by distinguishing several levels of uncertainty and inconsistency. Properties can then be interpreted over the *multi-valued* semantics [14, 13, 8]. When using a multi-valued semantics, the connection between abstract and concrete models (or between two abstract models) is usually model specific. This issue is discussed in [43], where the authors define (bi)simulation between two multi-valued models. There, the relation returns a value, indicating how "close" the models are. The "closeness" between the models refers to the universal fragment of μ-calculus. Our mixed simulation, on the other hand, returns either true or false, indicating whether one model is an abstraction of the other. Based on mixed simulation and on bilattices we extend the notion of preserving properties defined in some temporal logic (specifically, full μ-calculus formulas).

Mixed simulation for the 3-valued semantics is presented in [22, 29], which has the property of preserving the full μ-calculus. In [3] the authors present a mixed simulation relation for a 6-valued semantics. Their mixed simulation preserves temporal logic formulas consisting of both universal and existential quantifiers. The preservation of formulas is defined with respect to information order. Our work generalizes the notion of mixed simulation for any multi-valued semantics. Similar to [3], preservation of formulas is defined with respect to information order.

The abstraction is sometimes too coarse, which results in an inconclusive model checking result. In this case, the abstract model should be refined. *Refinement* is done be adding more details into the model, thus making it more similar to the concrete model. Refinement is traditionally done by splitting abstract states based on some criterion. The process of creating an abstract model, model checking it, and refining it iteratively is referred to as *abstraction-refinement*.

The traditional abstraction-refinement framework, referred to as CEGAR [44, 17] considers a 2-valued abstraction, which is conservative for *true*. Thus, $false$ may be a false-alarm, and refinement is then aimed at eliminating false results. The framework is designed for universal temporal logics, and is less suitable for temporal logics with both universal and existential operators, such as the full μ-calculus.

Several works investigate abstraction-refinement algorithms for temporal logics that combine both existential and universal quantifiers. Most of these works deal with either the 2-valued semantics or the 3-valued semantics. [45, 47, 2] suggest abstraction-refinement mechanisms for the μ-calculus over 2-valued semantics, for *specific* abstractions. In [51, 52] an automatic refinement mechanism, which is based on finding a *failure cause*, is presented. [3] suggest an abstraction-refinement framework for a specific 6-valued semantics.

In our work we generalize the discussion to abstract models over *any* multi-valued semantics, and present an abstraction-refinement algorithm for specifications in the full μ-calculus.

Compositional Verification

Another promising approach for fighting the state explosion problem is *compositional* model checking, where parts of the system are verified separately in order to avoid the construction of the entire system and to reduce the model checking cost. Evidence of the importance of the compositional approach can be found in the special issue of the journal Formal Methods in System Design, dedicated to the subject [28].

Usually, it is impossible to verify a component of the system in complete isolation from the rest of the system. This is because the behavior of one component depends on the interaction it has with its environment. To take into account dependencies between system components, the Assume-Guarantee (AG) paradigm [41, 49, 33] suggests how to verify one module based on an assumption on the behavior of its environment, where the environment consists of the other system components. The environment is then verified in order to guarantee that the assumption is actually satisfied.

Learning [1] has been a major technique to construct assumptions for the AG paradigm automatically. The *learning-based AG framework* is first described in [20]. It uses iterative AG reasoning, where in each iteration the assumption is modified based on the learning algorithm. Many works suggest optimizations of the basic framework and apply it in the context of different AG rules ([9, 27, 55, 25, 46, 34, 19, 48, 16]). These works are all designed for universal safety properties, with the exception of [25], which learns the full class of ω-regular languages.

[5] proposes an alternative approach for AG style compositional verification. It presents automated assume-guarantee reasoning by abstraction-refinement. There, the assumptions are computed as an over-approximating abstraction of the interface behavior of one of the components. When the abstraction is too coarse, refinement is done in a manner similar to CEGAR [17].

In contrast to approaches based on the AG paradigm, the compositional technique presented in [54] is based on a *3-valued abstraction*. There, the property is evaluated on each component separately with respect to the 3-

valued semantics. The result of the evaluation is a 3-valued marking of the states of the component. If the evaluation on the initial state of any component results in a definite value (true or false), then the same value holds for the composed system and the verification terminates. However, the evaluation may result in an indefinite value. In this case, only those component states where indefinite results have been obtained are identified and composed. As a result of using 3-valued abstraction, no false-negative or false-positive is obtained. This framework can verify and falsify the full μ-calculus.

To the best of our knowledge, all works which present a compositional approach deal with either 2-valued or 3-valued models. As opposed to that, we present a compositional framework to handle multi-valued models. Our framework generalizes the compositional approach in [54].

1.2 Organization

The rest of the thesis is organized as follows. In the next chapter we give the necessary background for multi-valued models, including the multi-valued μ-calculus, and a multi-valued model checking algorithm [53]. In Chapter 3 we describe bilattices, define Consistent Partial Distributive Bilattices and present their attributes. We then use bilattices as the basis for presenting mixed simulation for multi-valued models in Chapter 4. This sets the ground for describing the connection between a concrete model and its abstraction and for refinement. A refinement algorithm is presented in Chapter 4 as well. Our abstraction-refinement algorithm is based on the model checking graph.

Once abstraction-refinement for multi-valued models is defined, we can then present our compositional framework. We first investigate properties of partial model checking graphs in Chapter 5. Partial model checking graphs will later be used as part of our compositional framework. We then continue to describe composition of multi-valued models, and present our compositional framework in Chapter 6. Our compositional framework deals with the general case where the model which we want to verify is both an abstract model, and a composition of models. As the models at hand might be abstract, our compositional framework includes use of the abstraction-refinement algorithm presented in Chapter 4.

Both the abstraction-refinement algorithm, and the compositional framework presented in Chapters 4-6 deal with models defined over a CPDB. In Chapter 7 we discuss our framework with respect to full bilattices. Finally, we present our conclusions in Chapter 8.

Chapter 2

Preliminaries

In this chapter we introduce the concepts of lattices, multi-valued Kripke models, μ-calculus and multi-valued model checking graphs.

Definition 2.1 *Let $(L, \leq)$ be a partially ordered set. A subset B of L has a* least upper bound *$a \in L$, if the following holds:*

- *For every $b \in B$, $b \leq a$.*
- *If there exists $a' \in L$ such that for every $b \in B$, $b \leq a'$, then $a \leq a'$.*

Definition 2.2 *Let $(L, \leq)$ be a partially ordered set. A subset B of L has a* greatest lower bound *$a \in L$, if the following holds:*

- *For every $b \in B$, $a \leq b$.*
- *If there exists $a' \in L$ such that for every $b \in B$, $a' \leq b$, then $a' \leq a$.*

Definition 2.3 *A* lattice *$\mathcal{L}$=$(L, \leq)$ consists of a set L with a partial order $\leq$ over L, where every finite subset B of L has a least upper bound,* join, *denoted $\sqcup B$, and a greatest lower bound,* meet, *denoted $\sqcap B$, both in L. A lattice is* distributive *if $\sqcup$ and $\sqcap$ distribute over each other. That is, $(a \sqcap b) \sqcup c = (a \sqcup c) \sqcap (b \sqcup c)$, and $(a \sqcup b) \sqcap c = (a \sqcap c) \sqcup (b \sqcap c)$.*

Examples of lattices are shown in Figure 3.1(a),(b),(c),(e),(g) and (h).

Definition 2.4 *A* De Morgan algebra *is a structure $\mathcal{D}$=$(L, \leq, \neg)$, where $(L, \leq)$ is a finite distributive lattice, $\neg : L \rightarrow L$ is a negation function that satisfies for each a,b: $\neg\neg a = a$, $a \leq b \Leftrightarrow \neg b \leq \neg a$, and De Morgan laws are satisfied: $\neg(a \sqcap b) = \neg a \sqcup \neg b$, and $\neg(a \sqcup b) = \neg a \sqcap \neg b$.*

All De Morgan algebras have a greatest (top) element, denoted $true$, and a least (bottom) element, denoted $false$.

Definition 2.5 *Let $(L, \leq)$ be a partially ordered set. Function $f : L^n \rightarrow L$ is monotone with respect to $\leq$ iff $a_1 \leq a'_1, \ldots, a_n \leq a'_n$ implies $f(a_1, .., a_n) \leq f(a'_1, ..., a'_n)$.*

Definition 2.6 *Let $(L, \leq)$ be a partially ordered set. Function $f : L^n \rightarrow L$ is anti-monotone with respect to $\leq$ iff $a_1 \leq a'_1, \ldots, a_n \leq a'_n$ implies $f(a'_1, .., a'_n) \leq f(a_1, ..., a_n)$.*

2.1 Multi-Valued Models and μ-calculus

Definition 2.7 *A* Multi-Valued Kripke model *is a 6-tuple $M = \langle \mathcal{L}, AP, S, s_0, R, \Theta \rangle$, where:*

- *$\mathcal{L}$= $(L, \leq, \neg)$ - a De Morgan algebra,*
- *AP - a set of atomic propositions,*
- *S - a finite set of states,*
- *s_0 - the initial state,*
- *$R : S \times S \rightarrow L$ - a mapping of transitions to values in L,*
- *$\Theta : AP \rightarrow (S \rightarrow L)$ - a mapping which associates with each atomic proposition p, a mapping from S to L, describing the truth value of p in each state.*

Definition 2.8 *Let AP be a set of atomic propositions and Var a set of propositional variables. We consider the logic μ-calculus in negation normal form, defined as follows:*

$$\varphi \ ::= \ p \mid \neg p \mid Z \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \Box\varphi \mid \Diamond\varphi \mid \mu Z.\varphi \mid \nu Z.\varphi$$

where $p \in AP$ and $Z \in Var$.

Let L_μ denote the set of all formulas generated by the above grammar. Fixpoint quantifiers μ and ν are variable binders. We write η for either μ or ν. We assume formulas are well-named, i.e. no variable is bound more than once in any formula. Thus for a *closed* formula $\varphi \in L_\mu$, every variable Z

identifies a unique subformula $fp(Z) = \eta Z.\psi$ of φ. The set $Sub(\varphi)$ includes all subformulas of φ.

An *environment* $\mathcal{V} : Var \to (S \to L)$ defines the meaning of free variables. For a variable $Z \in Var$ and a mapping $l : S \to L$, we write $\mathcal{V}[Z = l]$ for the environment that agrees with $\mathcal{V}$ except that it maps Z to l.

We consider the multi-valued semantics of μ-calculus with respect to a multi-valued Kripke model M and an environment $\mathcal{V}$ [8]. Given such M and $\mathcal{V}$, a formula φ induces a mapping $S \to L$, denoted $\|\varphi\|^M_{\mathcal{V}}$, in which each state s of M is mapped to a value in L describing the truth value of φ in s. In the following, lfp, gfp stand for least and greatest fixpoints respectively, which exist based on [56].

The interpretation $\|\varphi\|^M_{\mathcal{V}}$ of $\varphi \in L_\mu$ in a multi-valued Kripke model M and environment $\mathcal{V}$ is defined as follows:

$$\begin{aligned}
\|p\|^M_{\mathcal{V}} &= \lambda s.\Theta(p)(s) \\
\|\neg p\|^M_{\mathcal{V}} &= \lambda s.\neg\Theta(p)(s) \\
\|\varphi_1 \vee \varphi_2\|^M_{\mathcal{V}} &= \lambda s.\|\varphi_1\|^M_{\mathcal{V}} \sqcup \|\varphi_2\|^M_{\mathcal{V}} \\
\|\varphi_1 \wedge \varphi_2\|^M_{\mathcal{V}} &= \lambda s.\|\varphi_1\|^M_{\mathcal{V}} \sqcap \|\varphi_2\|^M_{\mathcal{V}} \\
\|\Diamond\varphi\|^M_{\mathcal{V}} &= \lambda s. \bigsqcup_{s' \in S} (R(s, s') \sqcap \|\varphi\|^M_{\mathcal{V}}(s')) \\
\|\Box\varphi\|^M_{\mathcal{V}} &= \lambda s. \bigsqcap_{s' \in S} (\neg R(s, s') \sqcup \|\varphi\|^M_{\mathcal{V}}(s')) \\
\|Z\|^M_{\mathcal{V}} &= \mathcal{V}(Z) \\
\|\mu Z.\varphi\|^M_{\mathcal{V}} &= \mu(\lambda g.\|\varphi\|^M_{\mathcal{V}[Z=g]}) \\
\|\nu Z.\varphi\|^M_{\mathcal{V}} &= \nu(\lambda g.\|\varphi\|^M_{\mathcal{V}[Z=g]})
\end{aligned}$$

For closed formulas we drop the environment, and refer to $\|\varphi\|^M$. If M is clear from the context, we refer to $\|\varphi\|_{\mathcal{V}}$.

Definition 2.9 *Approximants of L_μ formulas are defined in the usual way: if $fp(Z) = \mu Z.\varphi$ then $Z^0 := \lambda s.false$, $Z^{i+1} := \|\varphi\|_{\mathcal{V}[Z=Z^i]}$ for any $i \in \mathbb{N}$ and any $\mathcal{V}$. Dually, if $fp(Z) = \nu Z.\varphi$, $Z^0 := \lambda s.true$ and $Z^{i+1} := \|\varphi\|_{\mathcal{V}[Z=Z^i]}$.*

Note that for every $i \in \mathbb{N}$, if $fp(Z) = \eta Z.\varphi$ then, by abuse of notation, Z^i can be viewed as a formula. That is, $Z^0 = false(true)$ (depending on η), and $Z^{i+1} = \|\varphi[Z = Z^i]\|_{\mathcal{V}}$, where $\varphi[Z = Z^i]$ represents the syntactic assignment of the formula Z^i instead of every occurrence of Z in φ. Thus, if $\psi \in Sub(\varphi)$, $\|\psi\|_{\mathcal{V}[Z=Z^i]} = \|\psi[Z = Z^i]\|_{\mathcal{V}}$, and Z is not free in $\psi[Z = Z^i]$.

Theorem 2.10 *[56] Given a Kripke model M with a finite state set S, for every μ-calculus formula φ there exists $\alpha \in \mathbb{N}$ such that for every $s \in S$, $\|\eta Z.\varphi\|_{\mathcal{V}}^{M}(s) = Z^{\alpha}(s)$.*

2.2 Multi-Valued Model-Checking Algorithm

A multi-valued model checking algorithm for a closed L_μ formula over a multi-valued Kripke model is suggested in [53]. There, multi-valued games are introduced, and a multi-valued model checking problem is translated into a problem of finding the value of a multi-valued game.

In this work, we only use the model-checking graph (further referred to as *mc-graph*) defined in [53], with its connections to the model checking algorithm.

Let $M = \langle \mathcal{L}, AP, S, s_0, R, \Theta \rangle$ be a multi-valued Kripke model over $\mathcal{L}= (L, \leq, \neg)$ and let φ_0 be a closed L_μ formula. The *mc-graph* is defined by $G(M, \varphi_0) = (n^0, N, E)$, where N is a set of nodes, $E \subseteq N \times N$ is the set of edges in the graph and $n^0 \in N$ is the initial node.

Nodes in the mc-graph are elements of $S \times Sub(\varphi_0)$, denoted $t \vdash \psi$ and $n^0 = s_0 \vdash \varphi_0$. Nodes are divided into $\vee$-nodes and $\wedge$-nodes. Nodes of type $s \vdash \varphi_0 \vee \varphi_1$ or $s \vdash \Diamond\varphi$ are $\vee$-nodes, whereas nodes of type $s \vdash \varphi_0 \wedge \varphi_1$ or $s \vdash \Box\varphi$ are $\wedge$-nodes. Nodes of type $s \vdash Z$ or $s \vdash \eta Z.\varphi$ can be either $\vee$-nodes or $\wedge$-nodes. The edges of the mc-graph are defined by the following rules.

$$\frac{s \vdash \varphi_0 \vee \varphi_1}{s \vdash \varphi_i}\ i \in \{0,1\} \qquad \frac{s \vdash \varphi_0 \wedge \varphi_1}{s \vdash \varphi_i}\ i \in \{0,1\}$$

$$\frac{s \vdash \Diamond\varphi}{t \vdash \varphi}\ R(s,t) \neq false \qquad \frac{s \vdash \Box\varphi}{t \vdash \varphi}\ R(s,t) \neq false$$

$$\frac{s \vdash \eta Z.\varphi}{s \vdash Z} \qquad \frac{s \vdash Z}{s \vdash \varphi}\ \text{if } fp(Z) = \eta Z.\varphi$$

Every edge $(n, n') \in E$ corresponds to a rule where n, n' are of the form of the upper, respectively lower, part of the rule. If no rule is defined from some node n, then there are no outgoing edges from n in the mc-graph. This happens in terminal nodes of the form $t \vdash p$ or $t \vdash \neg p$, or in terminal nodes of the form $t \vdash \Diamond\varphi$ or $t \vdash \Box\varphi$ where there are no transitions from the state t in the Kripke model.

Each edge in E is associated with a value from L: edges that refer to a transition of the model get the value of that transition. The rest get the value *true*. By abuse of notation we use $R(n, n')$ to refer to the value of an edge $(n, n') \in E$.

Definition 2.11 *([53]) Let n be a terminal node in G, $val(n)$ is defined as follows. $val(t \vdash q) = \Theta(q)(t)$, $val(t \vdash \neg q) = \neg\Theta(q)(t)$, $val(t \vdash \Diamond\varphi) = false$ and $val(t \vdash \Box\varphi) = true$.*

In [53] an algorithm for computing a value of nodes on a mc-graph is presented. The algorithm handles the *alternation-free* fragment of L_μ, where no nesting of fixpoints is allowed. Given a mc-graph $G(M, \varphi_0) = (n^0, N, E)$ and a function $val : N \to L$ which maps terminal nodes in G to values in L (Definition 2.11), the algorithm returns a mc-function $\chi : N \to L$ that maps each node to a value from L.

Algorithm 2.12 (mc-algorithm [53]) *G is partitioned to* Maximal Strongly Connected Components *(MSCCs) and a (total) order on them is determined, reflected by their numbers: Q_1,...,Q_k. The order fulfills the rule that if $i < j$ then there are no edges from Q_i to Q_j. The components are handled by increasing values of i. Consider a single Q_i. Each node $n \in Q_i$ is associated with a value $\chi(n)$ as follows.*
For a terminal node n, $\chi(n) = val(n)$.
For a $\vee$-node n we set $\chi(n)$ to be $\bigvee\{R(n, n') \wedge \chi(n') | R(n, n') \neq false\}$. Similarly, if n is a $\wedge$-node then $\chi(n) = \bigwedge\{\neg R(n, n') \vee \chi(n') | R(n, n') \neq false\}$.
If Q_i is a non-trivial MSCC then it contains exactly one fixpoint variable Z. In this case, first label the nodes in Q_i with temporary values, $temp(n)$, that are updated iteratively. For nodes of the form $n = s \vdash Z$, initialize $temp(n)$ to true if Z is of type ν, or to $false$ if Z is of type μ (the rest remain uninitialized). Then apply the previous rules for $\vee$,$\wedge$-nodes until the temporary values do not change anymore. Finally, set $\chi(n) = temp(n)$ for every node n in Q_i. Return χ as the mc-function.

In [53], the connection between χ and the model checking problem is proved, by showing that $\chi(n^0) = \|\varphi_0\|^M(s_0)$. In the context of this work we will also be interested in the internal nodes of G. We therefore generalize the correspondence between χ and the multi-valued semantics to *all* nodes in G.

For $\psi \in Sub(\varphi_0)$, we use ψ^* to denote the result of replacing every free occurrence of $Z \in Var$ in ψ by $fp(Z)$. Note that, ψ^* is a closed formula, and if ψ is closed then $\psi^* = \psi$.

Theorem 2.13 *Let $G(M, \varphi_0)$ be a mc-graph, such that φ_0 is an alternation-free closed L_μ formula. Let χ be the mc-function returned by the* ***mc-algorithm****, then for every $s \vdash \psi \in N$, $\chi(s \vdash \psi) = \|\psi^*\|^M(s)$.*

Proof: For some node $n = s \vdash \psi \in N$, the subgraph induced by n is the subgraph that includes all sons of n, recursively.

Since the valuation of the mc-graph is done bottom-up, and by the construction of the mc-graph, we can conclude that χ is correct for all nodes $n = s \vdash \psi$ such that ψ is a closed L_μ formula ([53]).

We now consider subformulas which are not closed. Let $\psi \in Sub(\varphi_0)$ be a L_μ formula. Let $G'(M, \psi^*)$ be the mc-graph created for the *closed* L_μ formula, ψ^*. Let χ' be the mc-function of G' (based on the **mc-algorithm**). We show that $\chi(s \vdash \psi) = \chi'(s \vdash \psi^*)$. If we look at G' and at the subgraph of G induced by n, the only difference between these graphs is an additional transitions in G' from nodes of the form $s' \vdash fp(Z)$ to node of the form $s' \vdash Z$. According to the algorithm, these transitions do not change the value of all the rest of the nodes. In particular, $\chi'(s \vdash \psi^*) = \chi(s \vdash \psi)$.

□

Let $G(M, \varphi_0)$ be a mc-graph. We say that $\chi : N \rightarrow L$ is *semantically correct* if for every $s \vdash \psi \in N$, $\chi(s \vdash \psi) = \|\psi^*\|^M(s)$.

Chapter 3

Bilattices and Partial Bilattices

In this chapter we introduce bilattices, consider several of their attributes, and define the notion of partial bilattices. The use of bilattices (or partial bilattices) for defining a multi-valued structure will then help us to describe a relation between two multi-valued Kripke models. The definition of such a relation is the basis for describing an abstraction-refinement algorithm, and for the compositional verification framework.

3.1 Bilattices and Partial Bilattices

Definition 3.1 *[26] A distributive bilattice is a structure* $\mathcal{B}$=$(B, \leq_i, \leq_t, \neg)$ *such that:*

1. $\mathcal{B}_i$=$(B, \leq_i)$ *is a lattice and* $\mathcal{B}_t$=$(B, \leq_t, \neg)$ *is a De Morgan algebra.*
2. *meet*($\otimes$) *and join*($\oplus$) *of* $\mathcal{B}_i$*, and meet*($\wedge$) *and join*($\vee$) *of* $\mathcal{B}_t$ *are monotone with respect to both* $\leq_i$ *and* $\leq_t$*.*
3. *all meets and joins distribute over each other.*
4. *negation* ($\neg$) *is* $\leq_i$ *monotone.*

Note that requirement (4) defines the monotonicity of $\neg$ with respect to $\leq_i$, whereas $\neg$ is anti-monotone with respect to $\leq_t$ (this is a result of $\mathcal{B}_t$ being a De Morgan algebra).

The bilattices considered in this work are distributive, thus the use of the term bilattice refers to distributive bilattice. In our context, the relation

$\leq_t$ is an order on the "degree of truth". The bottom in this order is denoted by $false$ and the top by $true$. Thus $false \leq_t x \leq_t true$ for any $x \in B$. The meet and join operations for $\leq_t$ are denoted by $\wedge$ and $\vee$ respectively. The relation $\leq_i$ is an order on the "degree of information". Thus, if $x \leq_i y$, y gives us at least as much information as x (and possibly more). The meet and join operations for $\leq_i$ are denoted $\otimes$ and $\oplus$ respectively. The bottom in the $\leq_i$ order is denoted by $\perp$ and the top by $\top$.

Definition 3.2 *[26] Let $\mathcal{D}$=$(D, \leq, \neg)$ be a De Morgan algebra. The* bilattice induced by *$\mathcal{D}$, denoted $\mathcal{B}(\mathcal{D})$, is a structure $(D \times D, \leq_i, \leq_t, \neg)$ such that:*

- $\langle a, b\rangle \leq_i \langle c, d\rangle \triangleq a \leq c$ *and* $b \leq d$
- $\langle a, b\rangle \leq_t \langle c, d\rangle \triangleq a \leq c$ *and* $d \leq b$
- $\neg\langle a, b\rangle \triangleq \langle b, a\rangle$

Theorem 3.3 *[26] Let $\mathcal{B}(\mathcal{D})$ be the bilattice induced by some De Morgan algebra $\mathcal{D}$. Then, $\mathcal{B}(\mathcal{D})$ is a distributive bilattice. Furthermore, every distributive bilattice is isomorphic to $\mathcal{B}(\mathcal{D})$ for some De Morgan algebra $\mathcal{D}$.*

Intuitively, for a De Morgan algebra $\mathcal{D}$, an element $\langle x, y\rangle$ of $\mathcal{B}(\mathcal{D})$ is interpreted as a value whose "degree of truth" is x and "degree of falsity" is y. If we view $\mathcal{D}$ as a concrete truth domain, $\mathcal{B}(\mathcal{D})$ can be viewed as its abstract truth domain. Given an element $c \in D$, $\langle x, y\rangle \in D \times D$ approximates c if x is no more true than c, and y is no more false than c. Thus, $\langle c, \neg c\rangle$ is the best approximation of c, and $\langle x, y\rangle$ approximates c if $\langle x, y\rangle \leq_i \langle c, \neg c\rangle$. We say that $\langle x, y\rangle \in D \times D$ is *consistent* if $\langle x, y\rangle \leq_i \langle c, \neg c\rangle$ for some $c \in D$. By the defintion of $\leq_i$, $\langle x, y\rangle \leq_i \langle c, \neg c\rangle$ iff $x \leq c$ and $y \leq \neg c$. Since $\neg$ is anti-monotone in De Morgan algebras (Definition 2.4), then $\langle x, y\rangle \leq_i \langle c, \neg c\rangle$ iff $c \leq \neg x$ and $y \leq \neg c$. As a result, $\langle x, y\rangle$ is consistent iff $y \leq \neg x$ (similarly to the definition presented in [38]). We say that $\langle x, y\rangle \in D \times D$ is *definite* if $\langle c, \neg c\rangle \leq_i \langle x, y\rangle$ for some $c \in D$. Thus $\langle x, y\rangle$ is definite iff $y \geq \neg x$. If $\langle x, y\rangle \in D \times D$ is both definite and consistent, then $\langle x, y\rangle = \langle c, \neg c\rangle$ for some $c \in D$.

Example 3.4 *Figure 3.1(a),(b) present an example of the distributive bilattice for the 4-valued Belnap structure ([4]). This bilattice is isomorphic to the bilattice $\mathcal{B}(\mathcal{D})$ created from the 2-valued De Morgan algebra*

$\mathcal{D}$= $(\{T, F\}, \leq, \neg)$, where $F \leq T$, $\neg T = F$. Thus, $t \triangleq \langle T, F\rangle$, $f \triangleq \langle F, T\rangle$, $\top \triangleq \langle T, T\rangle$ and $\bot \triangleq \langle F, F\rangle$. t, f are best approximations of T, respectively F. $\top$, representing maximal degree of truth and falsity, is inconsistent. t,f and $\top$ are definite elements. $\bot$ is indefinite.
Figure 3.1(g),(h) present the distributive bilattice for a $\langle 2^{\{a,b\}}, 2^{\{a,b\}}\rangle$ structure. This bilattice is generated from the De Morgan algebra $\mathcal{D}$= $(2^{\{a,b\}}, \leq, \neg)$, where $\leq$ is interpreted as the set-inclusion order, and $\neg$ is set complementation relative to $\{a, b\}$.

When referring to a bilattice $\mathcal{B}$, we sometimes implicitly refer to the structure $\mathcal{B}(\mathcal{D})$ isomorphic to $\mathcal{B}$ (which exists by Theorem 3.3). In particular, we use '$\leq$' to denote the order on the elements in the De Morgan algebra $\mathcal{D}$ of $\mathcal{B}(\mathcal{D})$.

Corollary 3.5 *Let $\mathcal{B}(\mathcal{D})$=$(D \times D, \leq_i, \leq_t, \neg)$ be a distributive bilattice, for every $\langle a, b\rangle, \langle c, d\rangle \in \mathcal{B}(\mathcal{D})$ the following holds:*

- $\langle a, b\rangle \wedge \langle c, d\rangle = \langle a \sqcap c, b \sqcup d\rangle$
- $\langle a, b\rangle \vee \langle c, d\rangle = \langle a \sqcup c, b \sqcap d\rangle$
- $\langle a, b\rangle \otimes \langle c, d\rangle = \langle a \sqcap c, b \sqcap d\rangle$
- $\langle a, b\rangle \oplus \langle c, d\rangle = \langle a \sqcup c, b \sqcup d\rangle$

Proof: We will prove this for the operator $\wedge$, the proof for the rest of the operators is similar. Assume $\langle a, b\rangle \wedge \langle c, d\rangle = \langle e, f\rangle$. Thus, by definition of greatest lower bound the following holds:

- $\langle e, f\rangle \leq_t \langle a, b\rangle$
- $\langle e, f\rangle \leq_t \langle c, d\rangle$
- For every $\langle e', f'\rangle$ if $\langle e', f'\rangle \leq_t \langle a, b\rangle$ and $\langle e', f'\rangle \leq_t \langle c, d\rangle$ then $\langle e', f'\rangle \leq_t \langle e, f\rangle$

Observe the element $\langle a \sqcap c, b \sqcup d\rangle$. Since $\mathcal{D}$ is a De Morgan algebra then $a \sqcap c \leq a$ and $a \sqcap c \leq c$. Similarly, $b \leq b \sqcup d$ and $d \leq b \sqcup d$. Thus, by the definition of truth order on $\mathcal{B}(\mathcal{D})$, $\langle a \sqcap c, b \sqcup d\rangle \leq_t \langle a, b\rangle$ and $\langle a \sqcap c, b \sqcup d\rangle \leq_t \langle c, d\rangle$. We can then conclude that $\langle a \sqcap c, b \sqcup d\rangle \leq_t \langle e, f\rangle$.

By the definition of truth order on $\mathcal{B}(\mathcal{D})$, since $\langle e, f\rangle \leq_t \langle a, b\rangle$ and $\langle e, f\rangle \leq_t \langle c, d\rangle$, we conclude that $e \leq a$ and $e \leq c$. But since $\mathcal{D}$ is a De Morgan algebra, this means that $e \leq a \sqcap c$. Similarly we can conclude that $b \sqcup d \leq f$. We can then conclude that $\langle e, f\rangle \leq_t \langle a \sqcap c, b \sqcup d\rangle$.

Based on the above we conclude that $\langle e, f\rangle = \langle a \sqcap c, b \sqcup d\rangle$.

□

Definition 3.6 *$\mathcal{P}$= $(B, \leq)$ is a* partial lattice *if it is a lattice, except that join is not always defined. A* partial distributive bilattice *is a structure $\mathcal{P}$=$(B, \leq_i, \leq_t, \neg)$ defined similarly to a distributive bilattice (Definition 3.1), except that $\mathcal{P}_i$=$(B, \leq_i)$ is a partial lattice, and requirements (2) and (3) hold for join of $\mathcal{P}_i$ only if it is defined.*

Definition 3.7 *Let $\mathcal{B}(\mathcal{D})$=$\langle D\times D, \leq_t, \leq_i, \neg\rangle$ be a bilattice, and let $P \subseteq D \times D$ be the set of all consistent elements in $\mathcal{B}(\mathcal{D})$. Then $\mathcal{P}(\mathcal{B})$=$\langle P, \leq_t, \leq_i, \neg\rangle$ is the* consistent structure *induced by $\mathcal{B}(\mathcal{D})$, where $\leq_t$, $\leq_i$ and $\neg$ in $\mathcal{P}(\mathcal{B})$ are as in $\mathcal{B}(\mathcal{D})$, restricted to consistent elements.*

Note that in consistent structures we do not have $\top$, the top element in the information order. However, $\bot$, $true$ and $false$ always exist.

3.2 Attributes of Bilattices and Partial Bilattices

We first show that consistent elements are closed under $\wedge$, $\vee$, $\neg$ and $\otimes$.

Lemma 3.8 *Let $\mathcal{B}$=$\langle D \times D, \leq_t, \leq_i, \neg\rangle$ be a bilattice, and let $x, y \in D \times D$ be consistent elements, then $x \wedge y$, $x \vee y$, $\neg x$ and $x \otimes y$ are consistent as well.*

Proof: We prove this on each of the operators:

Let $\langle a, b\rangle, \langle c, d\rangle \in D \times D$ be consistent elements. Then, $b \leq \neg a$ and $d \leq \neg c$.

- $\neg\langle a, b\rangle$: $\langle a, b\rangle$ is consistent, thus $b \leq \neg a$. Since $\mathcal{D}$ is a De Morgan algebra we can conclude that $\neg b \geq \neg\neg a$, thus $a \leq \neg b$. This means that $\langle b, a\rangle$ is consistent as well.
- $v = \langle a, b\rangle \wedge \langle c, d\rangle$: By definition $v = \langle a, b\rangle \wedge \langle c, d\rangle = \langle a \sqcap c, b \sqcup d\rangle$. As $\sqcup$ is monotone with respect to $\leq$ we can conclude that $(b \sqcup d) \leq (\neg a \sqcup \neg c)$. By De Morgan, this means that $(b \sqcup d) \leq \neg(a \sqcap c)$, thus v is consistent.
- $v = \langle a, b\rangle \vee \langle c, d\rangle$: By definition $v = \langle a, b\rangle \vee \langle c, d\rangle = \langle a \sqcup c, b \sqcap d\rangle$. As $\sqcap$ is monotone with respect to $\leq$ we can conclude that $(b \sqcap d) \leq (\neg a \sqcap \neg c)$. By De Morgan, this means that $(b \sqcap d) \leq \neg(a \sqcup c)$, thus v is consistent.

- $v = \langle a, b\rangle \otimes \langle c, d\rangle$: By definition $v = \langle a, b\rangle \otimes \langle c, d\rangle = \langle a \sqcap c, b \sqcap d\rangle$. As $\sqcup$ is monotone with respect to $\leq$ we can conclude that $(b \sqcup d) \leq (\neg a \sqcup \neg c)$. By De Morgan, this means that $(b \sqcup d) \leq \neg(a \sqcap c)$. We also know that $(b \sqcap d) \leq (b \sqcup d)$, thus v is consistent.

□

We will now show that the consistent structure induced by some bilattice $\mathcal{B}$ is a partial distributive bilattice. This is formalized by the following theorem.

Theorem 3.9 *Let $\mathcal{B}$=$\langle D \times D, \leq_t, \leq_i, \neg\rangle$ be a bilattice, and let $\mathcal{P}(\mathcal{B})$=$\langle P, \leq_t, \leq_i, \neg\rangle$ be the consistent structure induced by it, then $\mathcal{P}(\mathcal{B})$ is a partial distributive bilattice.*

For the proof of Theorem 3.9, we need the following Lemma.

Lemma 3.10 *Let $\mathcal{B}$=$\langle D \times D, \leq_t, \leq_i, \neg\rangle$ be a bilattice, and let $\mathcal{P}(\mathcal{B})$=$\langle P, \leq_t, \leq_i, \neg\rangle$ be the consistent structure induced by it. Let $a, b \in D \times D$ be consistent values, then the values of $a \vee b$, $a \wedge b$, $a \otimes b$ and $\neg a$ on $\mathcal{B}$ and $\mathcal{P}(\mathcal{B})$ are the same. Also, if the value of $a \oplus b$ is defined on $\mathcal{P}(\mathcal{B})$, then it is equal to the value of $a \oplus b$ on $\mathcal{B}$ as well.*

Proof: According to Lemma 3.8, the values of $a \vee b$, $a \wedge b$, $a \otimes b$ and $\neg a$ on $\mathcal{B}$ are consistent. Since $\leq_i$, $\leq_t$ and $\neg$ for $\mathcal{P}(\mathcal{B})$ are the same as for $\mathcal{B}$ reduced to the consistent elements, and since P consists of all consistent elements in $D \times D$, then the values on these operators are the same for $\mathcal{B}$ and $\mathcal{P}(\mathcal{B})$.

For $a \oplus b$, by definition, if $a = \langle a_1, a_2\rangle$ and $b = \langle b_1, b_2\rangle$, then $a \oplus b = \langle a_1 \sqcup b_1, a_2 \sqcup b_2\rangle$. If this is a consistent value, and since $\mathcal{P}(\mathcal{B})$ includes all consistent elements, then this value will be the result on $\mathcal{P}(\mathcal{B})$. But this will also be the value of $a \oplus b$ on $\mathcal{B}$. Thus, the values on $\mathcal{P}(\mathcal{B})$ and on $\mathcal{B}$ are the same. □

We now return to the proof of Theorem 3.9.

Proof: We show that each of the requirements on partial distributive bilattice (Definition 3.6) hold on $\mathcal{P}(\mathcal{B})$:

1. We show that $\mathcal{P}_i$=$(P, \leq_i)$ is a partial lattice. That is, $\mathcal{P}_i$ consists of a set P with partial order $\leq_i$ over P, where every finite subset of P has a greatest lower bound, called meet. According to Lemma 3.8, if $a, b \in D \times D$ are consistent, then $a \otimes b$ is consistent as well. $a \otimes b$ is the greatest lower bound of a and b, and since P includes all consistent elements in $D \times D$, then every finite subset of P has a greatest lower bound.
2. We show that $\mathcal{P}_t$=$(P, \leq_t, \neg)$ is a De Morgan algebra. By Lemma 3.10, for $a, b \in P$, the result of $a \wedge b$, $a \vee b$ and $\neg a$ is the same on $\mathcal{P}(\mathcal{B})$ and $\mathcal{B}$, thus since $\mathcal{B}_t$=$(D \times D, \leq_t, \neg)$ is a De Morgan algebra, so is $\mathcal{P}_t$.
3. We show that meet($\otimes$) and join($\oplus$) of $\mathcal{P}_i$, and meet($\wedge$) and join($\vee$) of $\mathcal{P}_t$ are monotone with respect to both $\leq_i$ and $\leq_t$, if they are defined (join of $\mathcal{P}_i$ might be undefined). By Lemma 3.10, if these operations are defined, then their value on $\mathcal{P}(\mathcal{B})$ is equal to their value on $\mathcal{B}$. Also, based on the definition of a consistent structure, the orders $\leq_i$ and $\leq_t$ on $\mathcal{P}(\mathcal{B})$ are the same as on $\mathcal{B}$, reduced to consistent elements. Thus, the monotonicity applies here as well.
4. We show that all meets and joins distribute over each other. Again, by Lemma 3.10, if these operations are defined, then their value on $\mathcal{P}(\mathcal{B})$ is equal to their value on $\mathcal{B}$. Thus since they distribute over each other under $\mathcal{B}$, then they also distribute over each other under $\mathcal{P}(\mathcal{B})$.
5. We show that negation $(\neg)$ is $\leq_i$ monotone. By Lemma 3.8, for $a \in D \times D$ a consistent element, $\neg a$ is consistent as well, and since $\leq_i$ under $\mathcal{P}(\mathcal{B})$ is equal to its definition under $\mathcal{B}$, reduced to consistent elements, then negation is $\leq_i$ monotone under $\mathcal{P}(\mathcal{B})$, since it is monotone under $\mathcal{B}$.

□

We refer to consistent structures, which, by Theorem 3.9, are also partial distributive bilattices, as *consistent partial distributive bilattices* (CPDB). Note that for CPDBs, the set of maximal elements with respect to the information order is exactly the set of definite elements, all of the form $\langle c, \neg c\rangle$ for some $c \in D$. This is because for every consistent $\langle x, y\rangle$, $y \leq \neg x$ and therefore $\langle x, \neg x\rangle \geq_i \langle x, y\rangle$.

Another attribute which we will need is that definite values are closed under $\wedge$, $\vee$ and $\neg$, as formalized in the following theorem.

Theorem 3.11 *Let $\mathcal{B}= \langle B, \leq_t, \leq_i \neg\rangle$ be either a distributive bilattice or a CPDB, and let $a, b \in B$ be definite values. Then $a \wedge b$, $a \vee b$ and $\neg a$ are definite as well.*

Proof: For $\mathcal{B}$ which is a CPDB, if $a \in B$ is definite, then $a = \langle c, \neg c\rangle$, for some $c \in D$. Thus, for proving the theorem, it enough to prove the following.

Let $\mathcal{B}=\langle D \times D, \leq_t, \leq_i, \neg\rangle$ be a bilattice, and let $a, b \in D \times D$ be definite values. Then $a \wedge b$, $a \vee b$ and $\neg a$ are definite as well. Furthermore, if $a = \langle c, \neg c\rangle$ and $b = \langle d, \neg d\rangle$ for some $c, d \in D$, then $a \wedge b$, $a \vee b$ and $\neg a$ are $\langle e, \neg e\rangle$ for some $e \in D$.

We prove both parts according to the different operators:

- $v = \neg\langle a, b\rangle$: By definition of negation $v = \langle b, a\rangle$ (Definition 3.2). Since we assume $\langle a, b\rangle$ is definite, then $b \geq \neg a$. Based on De Morgan, $a \geq \neg b$, thus v is definite as well.
  For $v = \neg\langle a, \neg a\rangle$ we get $v = \langle \neg a, a\rangle$, which, clearly, is of the required form.
- $v = \langle a, b\rangle \wedge \langle c, d\rangle$: By definition $v = \langle a \sqcap c, b \sqcup d\rangle$. We assume $b \geq \neg a$ and $d \geq \neg c$, then since the $\sqcup$ operator is monotone with respect to $\leq$, $b \sqcup d \geq \neg a \sqcup \neg c$. By De Morgan $b \sqcup d \geq \neg(a \sqcap c)$, thus v is definite.
  For $v = \langle a, \neg a\rangle \wedge \langle c, \neg c\rangle$ we get $v = \langle a \sqcap c, \neg a \sqcup \neg c\rangle$, and by De Morgan, this means that $v = \langle a \sqcap c, \neg(a \sqcap c)\rangle$, which is of type $\langle e, \neg e\rangle$ for $e = a \sqcap c \in D$.
- $v = \langle a, b\rangle \vee \langle c, d\rangle$: By definition $v = \langle a \sqcup c, b \sqcap d\rangle$. We assume $b \geq \neg a$ and $d \geq \neg c$, then since the $\sqcap$ operator is monotone with respect to $\leq$, $b \sqcap d \geq \neg a \sqcap \neg c$. By De Morgan $b \sqcap d \geq \neg(a \sqcup c)$, thus v is definite.
  For $v = \langle a, \neg a\rangle \vee \langle c, \neg c\rangle$ we get $v = \langle a \sqcup c, \neg a \sqcap \neg c\rangle$, and by De Morgan, this means that $v = \langle a \sqcup c, \neg(a \sqcup c)\rangle$, which is of type $\langle e, \neg e\rangle$ for $e = a \sqcup c \in D$.

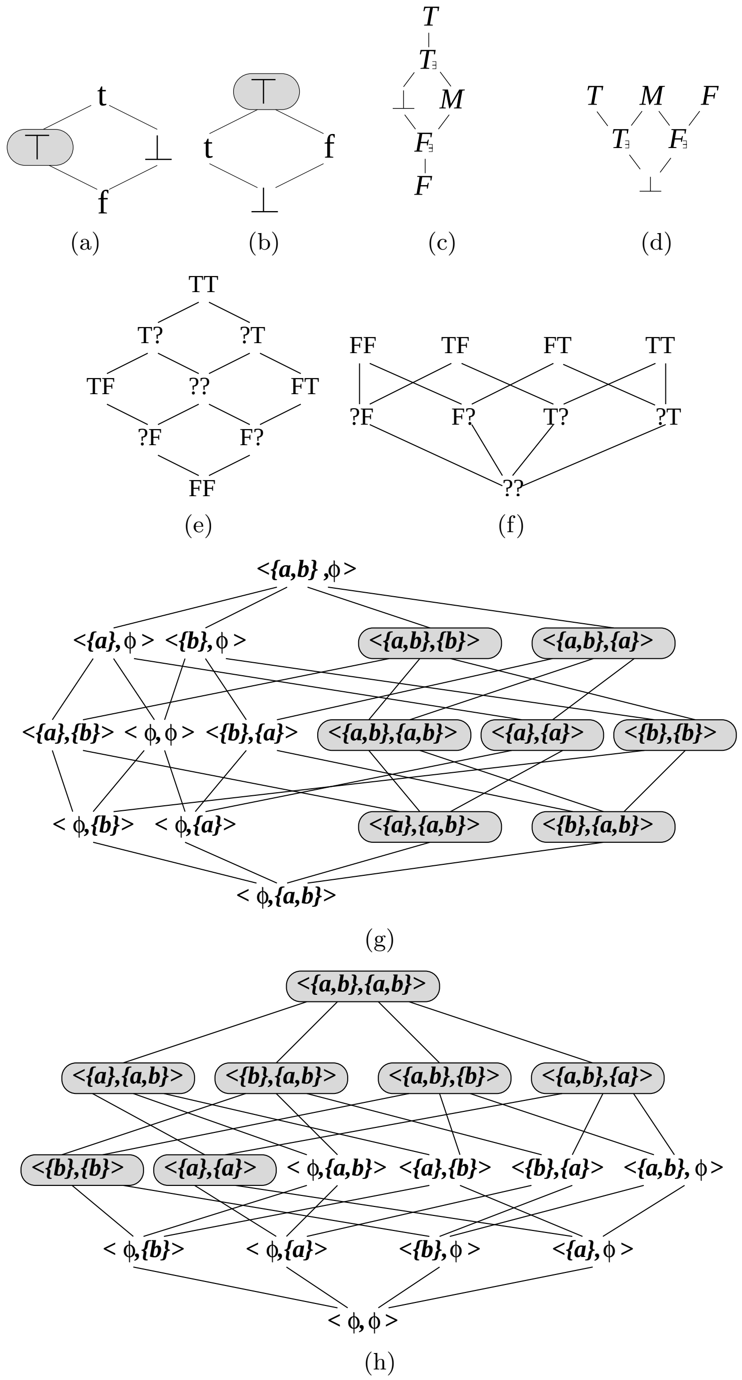
□

Example 3.12 *Examples of CPDBs appear in Figure 3.1. The CPDB induced by the bilattice of the Belnap structure is described in Figure 3.1(a) and (b), as the un-boxed elements, which are all the consistent elements. This CPDB is isomorphic to the standard 3-valued structure ([30]), where $? \triangleq \bot$, $T \triangleq t$ and $F \triangleq f$.*
The structure 3×3 is defined by the CPDB in Figure 3.1(e) and (f). This

CPDB is isomorphic to the CPDB induced by the bilattice $\mathcal{B}(\mathcal{D})$ created from the 2-views De Morgan algebra $\mathcal{D}= (\{T,F\} \times \{T,F\}, \leq, \neg)$, where $\leq$ and $\neg$ are defined bitwise. That is, for $a_1a_2, b_1b_2 \in \{T,F\} \times \{T,F\}$, $a_1a_2 \leq b_1b_2$ iff $a_1 \leq b_1$ and $a_2 \leq b_2$. Also, $\neg(a_1a_2) \triangleq \neg a_1 \neg a_2$. The 2-views De Morgan algebra is a special case of presenting inconsistent viewpoints ([24, 40]), when only 2 viewpoints are represented. That is, for $a_1a_2 \in \{T,F\} \times \{T,F\}$, a_1 represents the first view and a_2 represents the second view.

The 3×3 structure also represents two different views, which may be contradictory (e.g. TF). However, such elements should not be confused with inconsistent elements in $\mathcal{B}(\mathcal{D})$ such as $\langle TT, TT\rangle$. The consistent elements of $\mathcal{B}(\mathcal{D})$ are mapped into pairs over {T,F,?} in the 3×3 structure. E.g., $\langle TF, FF\rangle$ is represented by T? and $\langle TT, FF\rangle$ is represented by TT. The resulting structure contains both representations of the elements of the concrete 2-views domain (e.g. TT), and their approximations (e.g. T?).
The CPDB induced by the bilattice of the $\langle 2^{\{a,b\}}, 2^{\{a,b\}}\rangle$ structure is described in Figure 3.1(g) and (h), as the un-boxed elements, which are all the consistent elements. Note that the CPDB describing the 3×3 structure is isomorphic to the CPDB induced by the $\langle 2^{\{a,b\}}, 2^{\{a,b\}}\rangle$ structure. These two structures present a different way for presenting the same information. For example, the element $\langle\{a\},\{b\}\rangle$ is equivalent to the element TF in the sense that they both represent a similar evaluation. For the elements in the $\langle 2^{\{a,b\}}, 2^{\{a,b\}}\rangle$ structure, a represents the first view and b represents the second view. Thus both elements $\langle\{a\},\{b\}\rangle$ and TF represent the case of true by the first view and false by the second view. The use of a specific structure will depend, for example in this case, on the implementation. That is, we may represent an element of the logic as a vector (where its size depends on the number of views represented), or as two groups, a group of elements whose value is true and a group of elements whose value is false (the don't know elements are all the rest).

Multi-valued Kripke models as well as the semantics of L_μ formulas and mc-graphs are defined over a De Morgan algebra $\mathcal{L}$. These definitions can easily be extended to a *multi-valued structure*, which is either a distributive bilattice or a CPDB. Thus, we have both the information and truth lattices. In this case, the lattice $\mathcal{L}$ used in the multi-valued semantics is the truth lattice. This does not restrict the generality of our work since even if the multi-valued model is defined over a De Morgan algebra $\mathcal{L}$ which is not



(a) (b) (c) (d)

(e) (f)

(g)

(h)

Figure 3.1: Truth (a) and information (b) orders of 4-valued Belnap structure; Truth (c) and information (d) orders of 6-valued structure; Truth (e) and information (f) orders of 3×3 structure; Truth (g) and information (h) orders of $\langle 2^{\{a,b\}}, 2^{\{a,b\}} \rangle$ structure; Boxed nodes are inconsistent

(isomorphic to) a truth lattice of a bilattice (or a CPDB), then $\mathcal{L}$ can be lifted into $\mathcal{B}(\mathcal{L})$, resulting in a bilattice which contains both the original elements of $\mathcal{L}$ and their approximations. The model over $\mathcal{L}$ can then be interpreted as a model over $\mathcal{B}(\mathcal{L})$ or its CPDB, where each value of $\mathcal{L}$ is represented by its best approximation, and the semantics is maintained. For example, consider the 2-views De Morgan algebra which is lifted to a bilattice in Example 3.12.

In the rest of this work we use CPDBs. We will discuss this work with regards to full bilattices in Chapter 7.

Chapter 4

Mixed Simulation and Refinement of Multi-Valued Models

In this chapter we define a mixed simulation relation between two multi-valued Kripke models M_1 and M_2. Based on the mixed simulation relation, we describe the connection between abstract and concrete models. The mixed simulation relation also enables us to define the connection between two different abstractions of a model, where one is more precise than the other. Describing the connection between (abstract) models sets the ground for presenting a refinement algorithm. Our refinement algorithm, presented in Section 4.4, is based on the multi-valued model checking algorithm.

4.1 Multi-Valued Mixed Simulation

We first define a relation between two multi-valued Kripke models, both defined over the same multi-valued structure. This relation guarantees preservation of L_μ formulas with respect to the multi-valued semantics. The simulation relation is defined by means of the information order. Intuitively, it identifies the fact that M_2 contains less information than M_1. Thus, M_2 is an abstraction of M_1.

Definition 4.1 *Let* $M_1 = \langle \mathcal{L}, AP, S_1, s_0^1, R_1, \Theta_1 \rangle$ *and* $M_2 = \langle \mathcal{L}, AP, S_2, s_0^2,$

$R_2, \Theta_2\rangle$ be two multi-valued Kripke models. $H \subseteq S_1 \times S_2$ is a mixed simulation from M_1 to M_2 if $(s_1, s_2) \in H$ implies:

1. *For every $p \in AP$: $\Theta_2(p)(s_2) \leq_i \Theta_1(p)(s_1)$.*
2. *For every $t_1 \in S_1$ such that $R_1(s_1, t_1) \neq false$ there exists $t_2 \in S_2$ such that $(t_1, t_2) \in H$ and $R_2(s_2, t_2) \leq_i R_1(s_1, t_1)$.*
3. *For every $t_2 \in S_2$ such that $R_2(s_2, t_2) \not\leq_i false$ there exists $t_1 \in S_1$ such that $(t_1, t_2) \in H$ and $R_2(s_2, t_2) \leq_i R_1(s_1, t_1)$.*

If there is a mixed simulation H such that $(s_0^1, s_0^2) \in H$, then M_2 abstracts M_1, denoted $M_1 \preceq M_2$.

Note that requirements (2) and (3) are not symmetrical. By requirement (2), every transition in M_1 has a representation in M_2, whereas by requirement (3), only transitions in M_2 such that $R_2(s_2, t_2) \not\leq_i false$ have a representation in M_1. These requirements are similar to the requirements of mixed simulation in the 3-valued case ([29, 22]). There, every *may* transition in M_1 has a representation in M_2, and every *must* transition in M_2 has a representation in M_1. In the multi-valued case transitions which are may and not must are transitions for which $R(s, t) \leq_i false$.

Example 4.2 *Consider the models in Figure 4.1(a) and Figure 4.1(b). The underlying multi-valued structure is the 3×3 structure (described in Figure 3.1(e),(f)). The mixed simulation is given by:*
$H = \{(s_{00}, s_0), (s_{01}, s_0), (s_{10}, s_1), (s_{11}, s_1), (s_{20}, s_2), (s_{30}, s_3), (s_{31}, s_3)\}$. Since $(s_{00}, s_0) \in H$, we can conclude that $M_C \preceq M_A$.

Note that the transition from s_3 to s_1 in M_A does not have a matching transition in M_C. This complies with requirement (3) of the mixed simulation, as in the 3×3 structure, $false$ = FF, and F? $\leq_i$ FF. $R_A(s_3, s_1)$ = F?, thus by (3) there is no requirement for a matching transition.

The mixed simulation relation guarantees preservation of L_μ formulas, as formalized in the following theorem.

Theorem 4.3 *Let $H \subseteq S_1 \times S_2$ be a mixed simulation relation from M_1 to M_2, and let φ be a closed L_μ formula. Then for every $(s_1, s_2) \in H$, $\|\varphi\|^{M_2}(s_2) \leq_i \|\varphi\|^{M_1}(s_1)$.*

Proof: The proof is done by induction on the structure of the formula. In order to avoid handling the environment in subformulas, we will use the following formula transformation. Let $\varphi' \in Sub(\varphi)$ such that $\varphi' = \eta Z.\varphi_1(Z)$ for $\eta \in \{\mu, \nu\}$, and let t_1, t_2 be states in S_1, S_2 respectively. According to Theorem 2.10, there exists $\alpha_1 \in \mathbb{N}$ such that $\|\varphi'\|^{M_1}(t_1) = Z^{\alpha_1}(t_1)$. Similarly there exists $\alpha_2 \in \mathbb{N}$ such that $\|\varphi'\|^{M_2}(t_2) = Z^{\alpha_2}(t_2)$. Let $\alpha^* = max(\alpha_1, \alpha_2)$. Then $\|\varphi'\|^{M_1}(t_1) = Z^{\alpha^*}(t_1)$ and $\|\varphi'\|^{M_2}(t_2) = Z^{\alpha^*}(t_2)$. Note that Z^{α^*} is a new formula which does not include the variable Z.

For the given formula φ, we use the above transformation on all subformulas of type $\eta Z.\varphi_1(Z)$. This results in a new formula φ' for which $\|\varphi\|^{M_1}(t_1) = \|\varphi'\|^{M_1}(t_1)$ and $\|\varphi\|^{M_2}(t_2) = \|\varphi'\|^{M_2}(t_2)$. Furthermore, since φ is a closed L_μ, then φ' does not include any variables.

We will prove the theorem on the transformed φ', by induction on the structure of the formula.

Base: $\varphi = p \in AP$: $\|p\|^{M_2}(s_2) = \Theta_2(p)(s_2)$ and $\|p\|^{M_1}(s_1) = \Theta_1(p)(s_1)$. By definition of H as mixed simulation: $\Theta_2(p)(s_2) \leq_i \Theta_1(p)(s_1)$. Thus, $\|p\|^{M_2}(s_2) \leq_i \|p\|^{M_1}(s_1)$.

Step:

- $\varphi = \neg p$ where $p \in AP$:
  By definition, $\|\neg p\|^{M_2}(s_2) = \neg\Theta_2(p)(s_2)$ and $\|\neg p\|^{M_1}(s_1) = \neg\Theta_1(p)(s_1)$. By definition of H as mixed simulation, $\Theta_2(p)(s_2) \leq_i \Theta_1(p)(s_1)$. Since $\neg$ is monotone with respect to $\leq_i$, we can conclude that $\|\neg p\|^{M_2}(s_2) \leq_i \|\neg p\|^{M_1}(s_1)$.
- $\varphi = \varphi_1 \wedge \varphi_2$:
  By definition, $\|\varphi_1 \wedge \varphi_2\|^{M_i}(s_i) = \|\varphi_1\|^{M_i}(s_i) \wedge \|\varphi_2\|^{M_i}(s_i)$ for $i \in \{1, 2\}$. By the induction hypothesis, $\|\varphi_1\|^{M_2}(s_2) \leq_i \|\varphi_1\|^{M_1}(s_1)$ and $\|\varphi_2\|^{M_2}(s_2) \leq_i \|\varphi_2\|^{M_1}(s_1)$. The induction hypothesis holds since $\varphi_1, \varphi_2 \in Sub(\varphi_1 \wedge \varphi_2)$.
  Since $\wedge$ is monotone with respect to $\leq_i$, we can conclude that: $\|\varphi_1\|^{M_2}(s_2) \wedge \|\varphi_2\|^{M_2}(s_2) \leq_i \|\varphi_1\|^{M_1}(s_1) \wedge \|\varphi_2\|^{M_1}(s_1)$.
  Thus, $\|\varphi_1 \wedge \varphi_2\|^{M_2}(s_2) \leq_i \|\varphi_1 \wedge \varphi_2\|^{M_1}(s_1)$.
- $\varphi = \varphi_1 \vee \varphi_2$: Dual to the case of $\varphi = \varphi_1 \wedge \varphi_2$.
- $\varphi = \Box\varphi_1$:
  By definition, $\|\Box\varphi_1\|^{M_i}(s_i) = \bigwedge_{t_i \in S_i} \left(\neg R_i(s_i, t_i) \vee \|\varphi_1\|^{M_i}(t_i)\right)$ for $i \in \{1, 2\}$.

We first notice that proving $\bigwedge_{j=1}^{n} a_j \leq_i \bigwedge_{j=1}^{m} b_j$ is identical to proving $\bigwedge_{j=1}^{n} a_j \leq_i \bigwedge_{j=1}^{m} b_j \wedge true$. For proving that, since $\wedge$ is monotone with respect to $\leq_i$, it suffices to show that:

- $\forall j \exists k . a_j \leq_i b_k$ or $a_j \leq_i true$ and
- $\forall k \exists j . a_j \leq_i b_k$

Note that it does not suffice to prove only one of the requirements, since the conjunction is done based on the truth order, whereas both conjuncts are compared with respect to the information order.
We need to prove $\bigwedge_{t_2 \in S_2} \big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \bigwedge_{t_1 \in S_1} \big(\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1)\big)$.
We first prove that for all $t_2 \in S_2$, either $\big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_i true$ or there exists $t_1 \in S_1$ such that $\big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \big(\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1)\big)$.
We then prove that for all $t_1 \in S_1$ there exists $t_2 \in S_2$ such that $\big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \big(\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1)\big)$.
For each $t_2 \in S_2$:
If $R_2(s_2, t_2) \not\leq_i false$, then by the definition of mixed simulation, there exists $t_1 \in S_1$ such that $(t_1, t_2) \in H$ and $R_2(s_2, t_2) \leq_i R_1(s_1, t_1)$. As negation is monotone with respect to $\leq_i$, we have: $\neg R_2(s_2, t_2) \leq_i \neg R_1(s_1, t_1)$.
Since $(t_1, t_2) \in H$, then by the induction hypothesis: $\|\varphi_1\|^{M_2}(t_2) \leq_i \|\varphi_1\|^{M_1}(t_1)$.
Since $\vee$ is monotone with respect to $\leq_i$, then $\big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \big(\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1)\big)$.
For the case where $R_2(s_2, t_2) \leq_i false$, since negation is monotone with respect to $\leq_i$, then $\neg R_2(s_2, t_2) \leq_i true$.
Since $\vee$ is monotone with respect to $\leq_i$, we can conclude that: $\big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \big(true \vee \|\varphi_1\|^{M_2}(t_2)\big) = true$
Thus, we can conclude that for every $t_2 \in S_2$ at least one of the following is true:

- there exists $t_1 \in S_1$ such that $(t_1, t_2) \in H$ and $\big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \big(\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1)\big)$.
- $\big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_i true$

For each $t_1 \in S_1$:
If $R_1(s_1, t_1) \neq false$ then by the definition of mixed simulation, there

exists $t_2 \in S_2$ such that $(t_1, t_2) \in H$ and $R_2(s_2, t_2) \leq_i R_1(s_1, t_1)$.
Thus, again we can conclude that: $\big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \big(\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1)\big)$.
If $R_1(s_1, t_1) = false$, then $\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1) = true$, thus t_1 does not affect the value of $\bigwedge_{t_1 \in S_1} \big(\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1)\big)$.
This means that: $\bigwedge_{t_2 \in S_2} \big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \bigwedge_{t_1 \in S_1} \big(\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1)\big)$.
We conclude, $\|\Box\varphi_1\|^{M_2}(s_2) \leq_i \|\Box\varphi_1\|^{M_2}(s_1)$.

- $\varphi = \Diamond\varphi_1$:
  By definition, $\|\Diamond\varphi_1\|^{M_i}(s_i) = \bigvee_{t_i \in S_i} \big(R(s_i, t_i) \wedge \|\varphi_1\|^{M_i}(t_i)\big)$ for $i \in \{1, 2\}$.
  We first notice that proving $\bigvee_{j=1}^{n} a_j \leq_i \bigvee_{j=1}^{m} b_j$ is identical to proving $\bigvee_{j=1}^{n} a_j \leq_i \bigvee_{j=1}^{m} b_j \vee false$. For proving that, since $\vee$ is monotone with respect to $\leq_i$, it suffices to show that:

  - $\forall j \exists k. a_j \leq_i b_k$ or $a_j \leq_i false$ and
  - $\forall k \exists j. a_j \leq_i b_k$

  We need to prove $\bigvee_{t_2 \in S_2} \big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \bigvee_{t_1 \in S_1} \big(R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)\big)$.
  We first prove that for all $t_2 \in S_2$, either $\big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_i false$ or there exists $t_1 \in S_1$ such that $\big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \big(R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)\big)$.
  we then prove that for all $t_1 \in S_1$ there exists $t_2 \in S_2$ such that $\big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \big(R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)\big)$.
  For each $t_2 \in S_2$:
  If $R_2(s_2, t_2) \not\leq_i false$, then by the definition of mixed simulation, there exists $t_1 \in S_1$ such that $(t_1, t_2) \in H$ and $R_2(s_2, t_2) \leq_i R_1(s_1, t_1)$.
  Since $(t_1, t_2) \in H$, then by the induction hypothesis: $\|\varphi_1\|^{M_2}(t_2) \leq_i \|\varphi_1\|^{M_1}(t_1)$.
  Since $\wedge$ is monotone with respect to $\leq_i$, we conclude that $R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2) \leq_i R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)$.
  For the case where $R_2(s_2, t_2) \leq_i false$. Since $\wedge$ is monotone with respect to $\leq_i$, we can conclude that: $\big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \big(false \wedge \|\varphi_1\|^{M_2}(t_2)\big) = false$
  Thus, we can conclude that for every $t_2 \in S_2$ at least one of the following is true:

– there exists $t_1 \in S_1$ such that $(t_1, t_2) \in H$ and $\big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \big(R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)\big)$.

– $\big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_i false$

For each $t_1 \in S_1$:
If $R_1(s_1, t_1) \neq false$ then by the definition of mixed simulation, there exists $t_2 \in S_2$ such that $(t_1, t_2) \in H$ and $R_2(s_2, t_2) \leq_i R_1(s_1, t_1)$. Thus, again we can conclude that: $\big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \big(R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)\big)$.
If $R_1(s_1, t_1) = false$, then $R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1) = false$, thus t_1 does not affect the value of $\bigvee_{t_1 \in S_1} \big(R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)\big)$.
This means that: $\bigvee_{t_2 \in S_2} \big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_i \bigvee_{t_1 \in S_1} \big(R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)\big)$.
We conclude, $\|\Diamond\varphi_1\|^{M_2}(s_2) \leq_i \|\Diamond\varphi_1\|^{M_1}(s_1)$.

□

4.2 Concrete and Abstract Multi-Valued Models

Abstraction refers to removing or simplifying details of the original model in order to obtain a smaller (abstract) model. In an abstract model, an abstract state may represent 0 or more states in the concrete model. A transition from one abstract state to another represents the transitions between the corresponding concrete states. It is, however, possible to add transitions in the abstract model that do not represent concrete transitions. We will not discuss how abstract models are constructed. This is investigated for instance in [38].

A model is more abstract than another model if the prior model has less information than the latter one. Recall that when presenting distributive bilattices we discussed that given a De Morgan algebra $\mathcal{D}= (D, \leq, \neg)$, then for any element $c \in D$, we have that $\langle x, y\rangle \in D \times D$ approximates c if $\langle x, y\rangle \leq_i \langle c, \neg c\rangle$. Since $\langle c, \neg c\rangle$ is consistent, and all $\langle x, y\rangle$ where $\langle x, y\rangle \leq_i \langle c, \neg c\rangle$ are consistent as well, we can conclude that for any given element $c \in D$ all elements approximating c are in $\mathcal{P}(\mathcal{B}(\mathcal{D}))$. Thus, for a model M defined over $\mathcal{D}$, a model M_A, which is an abstraction of M, is defined over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$.

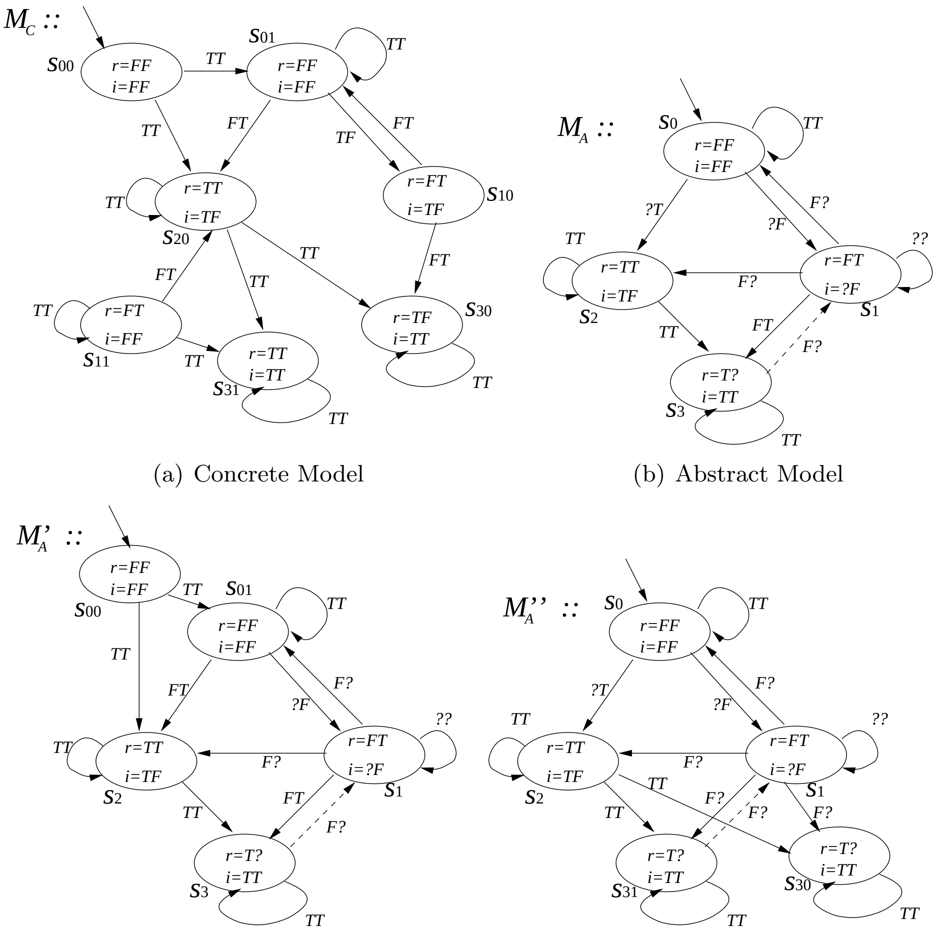
The mixed simulation relation is used to describe the relation between a model M defined over $\mathcal{D}$, and its abstraction, M_A: $M \preceq M_A$. This is since the mixed simulation relation captures the notion of one model including less information (being more abstract) than the other. Note, however, that when defining the mixed simulation relation both models are defined over the same multi-valued structure, whereas in our case M is defined over $\mathcal{D}$ and M_A is defined over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$. To overcome the above problem, we interpret M as a model over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$, instead of over $\mathcal{D}$. Note that M is defined over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$, in which all values are definite. Following Theorem 3.11, evaluating M over either $\mathcal{D}$ or over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$ will have the same results.

Resulting from the above, the mixed simulation relation can, indeed, be used to describe the relation between a model M and its abstraction M_A. The mixed simulation relation can also be used to describe the connection between two abstract models, both defined over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$, where one model is an abstraction of the other model.

We will sometimes refer to the model M, defined over $\mathcal{D}$, as the concrete model, and mark it M_C. We refer to this model as concrete since all its values are definite, and not approximated.

Example 4.4 *Consider the model M_C in Figure 4.1(a). The underlying multi-valued structure is the 3×3 structure (described in Figure 3.1(e),(f)). Figure 4.1(b) describes an abstraction of the model, $M_C \preceq M_A$ (the mixed simulation relation H is given in Example 4.2). Each (abstract) state s_A in M_A represents one or more states in M_C. These are the states which are in H with s_A. Thus, s_0 in M_A represents both s_{00} and s_{01} in M_C. Similarly, s_1 in M_A represents s_{10} and s_{11} in M_C. Note that indeed, s_1 includes less details than s_{10} and s_{11}, since the value of i is indefinite in s_1. The transition between s_0 and s_1 in M_A represents all transitions from s_{00} and s_{01} to s_{10} and s_{11} in M_C. Indeed, the value of the abstract transition from s_0 to s_1 is smaller by information order than the corresponding transitions.*

Note the dashed transition from s_3 to s_1 in M_A. This transition does not correspond to a transition in the concrete model M_C. This does not effect the mixed simulation between the models, as based on requirement 3 transitions whose value is less than or equal to $false$ by information order might not have a corresponding transition. Such transitions may still be created during the building of the abstract model due to abstraction which is not precise.



(a) Concrete Model

(b) Abstract Model

(c) Refined Abstract Model

(d) Another Refined Abstract Model

Figure 4.1: A concrete model M_C and its abstractions M_A and M'_A

4.3 Refinement of Models

Given an abstract model, the information order enables us to capture the notion of a model checking result being “not good enough”. This is a result that does not give us the maximal information possible. That is, it is indefinite. Such a result needs to be refined. On the other hand, a definite result is a result that gives us the maximal information possible.

Works such as [31, 32] present how to refine a model based on the mc-graph for 3-valued models. There, an indefinite result is accompanied with a failure cause. The failure cause is either an atomic proposition whose value is indefinite, or an outgoing transition whose value is indefinite. Refinement is then performed by splitting the abstract states in a way that eliminates the failure cause.

Similarly, our refinement is also performed by splitting states. The refinement consists of two parts. First, we choose a criterion for model refinement. Then, based on the criterion, the model is refined by splitting states. Splitting states is aimed at increasing the information level of some transition or of an atomic proposition in some state, based on the criterion. Note, however, that splitting states can, as a by-product, cause decreasing of the information level of some transitions.

Example 4.5 *Figure 4.1(c) describes a refinement of the abstract model M_A given in Figure 4.1(b). The refinement is done by state splitting: s_0 is split into s_{00} and s_{01}. Splitting these states enables us to increase the information level of the transition from s_0 to s_2. The value of this transition in M_A is $?T$, whereas after the refinement the transition is described by the two transitions from s_{00} to s_2 with value TT, and from s_{01} to s_2 with value FT.*

A different refinement of the abstract model M_A is given in Figure 4.1(d). The refinement is done by state splitting: s_3 is split into s_{30} and s_{31}. This state splitting causes decreasing of the information level of the transition from s_1 to s_3. In M_A this transition has the value FT. In the refined model, this transition is replaced with two transition from s_1 to both s_{30} and s_{31}, both with the value $F?$. Note that if the value of these transitions in M''_A had remained FT, then the mixed simulation relation between M_C and M''_A would not have been maintained.

4.4 Choosing Criterion for Refinement

In the rest of the section we study the first part of the refinement, which is choosing a criterion for model refinement. Consider a mc-graph $G = (n_0, N, E)$. For a mc-function $\chi : N \rightarrow L$, such that $\chi(n^0)$ is indefinite, our goal in the refinement is to find and eliminate at least one of the causes of the indefinite result. The criterion for the refinement is obtained from a *failure node* for n^0.

Definition 4.6 *Let $G = (n^0, N, E)$ be a mc-graph and let $\chi : N \rightarrow L$ be a mc-function on G such that $\chi(n)$ is indefinite. A node $n' = s \vdash \varphi \in N$ is a* failure node *for n if the following holds.*

1. *$\chi(n')$ is indefinite.*
2. *There is a path from n to n' on G, such that for every node n'' on the path, $\chi(n'')$ is indefinite.*
3. *$\chi(n')$ can be changed by either increasing the information level of an atomic proposition in s or by increasing the information level of some transition from s.*

Intuitively, if n is a failure node for n' then requirement 3 means that n *itself* is responsible for introducing (some) uncertainty. (1) and (2) require that this uncertainty is relevant to $\chi(n')$.

We adapt the **mc-algorithm** (Algorithm 2.12) to remember for each node whose value is indefinite a failure node and a failure reason. The failure node and reason of n^0 will be used as a criterion for refinement.

For every terminal node n, if $\chi(n)$ is indefinite, the failure node and reason attached to it are the node itself. To handle nonterminal nodes, we define an auxiliary function $f : N \rightarrow L$ that keeps for each node $n \in N$ its most updated value in the algorithm: If $\chi(n)$ is defined, then $f(n) = \chi(n)$. Otherwise, if *temp*(n) is defined, then $f(n) =$ *temp*(n).

Let n be a node for which $f(n)$ has been updated last. If $f(n)$ is definite, then no failure node and reason are attached to it. If $f(n)$ is indefinite, do the following:

1. If n is a $\vee$-node, find some node n' with $R(n, n') \neq false$ and for which the following requirements hold:
    (a) $\forall n'' \in N$ where $R(n, n'') \neq false$ one of the following holds:

        i. $(R(n,n'') \wedge f(n'')) \leq_t (R(n,n') \wedge f(n'))$. Or
        ii. $(R(n,n'') \wedge f(n''))$ and $(R(n,n') \wedge f(n'))$ are uncomparable.

    (b) $R(n,n') \wedge f(n')$ is indefinite.

   Intuitively, for some n', if requirement (a) holds then $R(n,n') \wedge f(n')$ is maximal, and thus affects $f(n)$. Requirement (b) ensures that it is possible to refine $R(n,n')$ or $f(n')$. For the given node n and a chosen node n' satisfying (a),(b) define a failure node and reason for n as follows:

   i If $f(n')$ is definite or $R(n,n') \lneq_t f(n')$: n is a failure node, and the edge (n,n') is the failure reason.

   ii If $R(n,n')$ is definite or $f(n') \lneq_t R(n,n')$, then the failure node and reason of n are those of n'.

   iii Otherwise, arbitrarily choose either n as a failure node and the edge as a failure reason, or the failure node and reason of n' as the failure node and reason of n.

2. The case where n is a $\wedge$-node is dual, where instead of searching for a maximal $R(n,n') \wedge f(n')$, we now try to find a minimal $\neg R(n,n') \vee f(n')$:
   If n is a $\wedge$-node, find some node n' with $R(n,n') \neq false$ and for which the following requirements hold:

    (a) $\forall n'' \in N$ where $R(n,n'') \neq false$ one of the following holds:
        i. $(\neg R(n,n') \vee f(n')) \leq_t (\neg R(n,n'') \vee f(n''))$. Or
        ii. $(\neg R(n,n') \vee f(n'))$ and $(\neg R(n,n'') \vee f(n''))$ are uncomparable.

    (b) $\neg R(n,n') \vee f(n')$ is indefinite.

   For the given node n and a chosen node n' satisfying (a),(b) define a failure node and reason for n as follows:

   i If $f(n')$ is definite or $f(n') \lneq_t \neg R(n,n')$, then n is a failure node, the edge (n,n') is the failure reason.

   ii If $R(n,n')$ is definite or $\neg R(n,n') \lneq_t f(n')$, then the failure node and reason of n are those of n'.

   iii Otherwise, arbitrarily choose either n as a failure node and the edge as a failure reason, or the failure node and reason of n' as the failure node and reason of n.

Definite values are closed under $\neg$, $\wedge$ and $\vee$ (Theorem 3.11), thus if a node is given an indefinite value, this indefinite value results from an indefinite value of either a son n' of n, or an edge from n. For example, consider case $1(i)$. If $f(n')$ is definite, then $R(n,n')$ is necessarily indefinite (Theorem 3.11). Similarly, if $R(n,n') \lneq_t f(n')$, then $R(n,n') \wedge f(n') = R(n,n')$, which again, means that $R(n,n')$ is indefinite. Either way, $R(n,n')$ can be refined and is therefore the failure reason. The correctness of the search for the failure node and reason is formalized by the following two lemmas.

Lemma 4.7 *For every nonterminal node n, if $f(n)$ is given an indefinite value, then there exists n' such that $R(n,n') \neq false$, which satisfies requirements (a),(b). Furthermore, $f(n')$ is already defined at that time. In addition, if the updating of failure node and reason of n is based on n', then n' also has a failure node and reason.*

Proof: We prove each part of the lemma separately.

First, we prove that if $f(n)$ is given an indefinite value, then $f(n')$ is defined. The updating of $f(n)$ follows the updating of *temp*(n) and $\chi(n)$. The only case where $f(n)$ is updated for a non-terminal node $n \in Q_i$, and updating does *not* depend on the sons of n, is when n is of the form $s \vdash Z$, for some variable Z, during the initialization of *temp* for Q_i. Based on the algorithm, in this case *temp*(n) is either *true* or $false$ (depending on the type of $fp(Z)$), and thus $f(n)$ is given a definite value. All other updates on internal node n are done based on the sons of n, thus are done only if either *temp*(n') or $\chi(n')$ are updated, for n' son of n. As a result, $f(n')$ is defined on n'.

Second, we prove that for every node n, if $f(n)$ is given an indefinite value, then there exists n' such that $R(n,n') \neq false$, which satisfies requirements (a),(b). We prove the lemma according to the type of the node n for which $f(n)$ is indefinite. We first consider the case of a $\vee$-node. We first show that only sons complying with requirement (a) influence $f(n)$. Assume a son n' of n that does not comply with requirement (a). Thus, there exists n'' which is a son of n for which $(R(n,n'') \wedge f(n''))$ and $(R(n,n') \wedge f(n'))$ are comparable. Furthermore, $(R(n,n') \wedge f(n')) \lneq_t (R(n,n'') \wedge f(n''))$. Recall that if $b \leq_t a$ then $a \vee b = a$. We conclude that n' does not influence $f(n)$. Therefore, only sons complying with requirement (a) influence $f(n)$.

Assume by way of contradiction that there does not exist a son n' which satisfies both requirements (a) and (b). Since all sons influencing $f(n)$ comply with requirement (a), we conclude that they all do not comply with requirement (b). Thus, for all sons influencing $f(n)$, $R(n, n') \wedge f(n')$ is definite. By Theorem 3.11, since all the values influencing $f(n)$ are definite, then $f(n)$ is definite as well. This contradicts the assumption that $f(n)$ is indefinite. We conclude that if $f(n)$ is indefinite, then there exists n' which satisfies requirements (a),(b).

Third, we prove that if the updating of failure node and reason on n is based on the failure node and reason of n', then a failure node and reason is defined on n'. By the definition of the failure node and reason, if the failure node and reason of n is based on the failure node and reason of n', then $f(n')$ is indefinite. By induction on the updating steps of $f(n)$, it is possible to prove that for all nodes, if $f(n)$ is indefinite, then it has a failure node and reason. □

A failure node and reason for n is updated *every time* $f(n)$ is updated. Thus, when the **mc-algorithm** terminates, for every n, if $\chi(n)$ is indefinite, then the failure node and reason for n is based on χ.

Lemma 4.8 *For every node n, if n is updated with a failure node n', then n' is a failure node for n (Definition 4.6).*

Proof: We prove by induction on the updating steps of $f(n)$ that for every n, if $f(n)$ is given an indefinite value and n is updated with a failure node $n' = s \vdash \varphi$ then the following holds:

1. $f(n')$ is indefinite.
2. There is a path from n to n' on G, such that for every node n'' on the path, $f(n'')$ is indefinite.
3. $f(n')$ can be changed by either increasing the information level of an atomic proposition in s or by increasing the information level of some transition from s.

Base: For terminal nodes, if $f(n)$ is indefinite, the failure node and reason attached to it are the node itself. Clearly, n complies with requirements 1-3.
Step: For nonterminal nodes, if $f(n)$ is given an indefinite value and updated with a failure node and reason n'. By Lemma 4.7 and based on the

updating algorithm, $f(n')$ is already defined for n'. By the hypothesis assumption, if $f(n')$ is indefinite then n' is updated with a failure node and reason $n'' = s'' \vdash \varphi''$, which complies with requirements 1-3.

The failure node and reason for n can be either those of n' or n is the failure node and the edge to n' is the failure reason.

If the failure node and reason is that of n', then the failure node and reason for n is n'' and the following is true:

1. Based on hypothesis assumption, $f(n'')$ is indefinite.
2. Based on the algorithm, $f(n')$ is indefinite. Based on the hypothesis assumption, there is a path from n' to n'' on G, such that for every node m on the path, $f(m)$ is indefinite. Thus, there is a path from n to n'' on G, such that from every node m on the path, $f(m)$ is indefinite.
3. By the hypothesis assumption, $f(n'')$ can be changed by either increasing the information level of an atomic proposition in s'' or by increasing the information level of some transition from s''.

If the failure node is n and the edge to n' is the failure reason, then the following holds:

1. Based on the algorithm, $f(n)$ is indefinite.
2. Trivially, there is a path from n to itself where $f(n)$ is indefinite.
3. Based on the algorithm, the value of the edge to n' influences $f(n)$ and is indefinite. Thus, $f(n)$ can be changed by increasing the information level of this transition.

The failure node and reason for n is updated *every time* $f(n)$ is updated. Thus, when the **mc-algorithm** terminates, for every n, if $\chi(n)$ is indefinite, then the failure node for n is indeed a failure node. □

Altogether there are two cases in which we consider the node itself as a failure node. The first case is when the node is a terminal node whose value is indefinite, for which the failure reason is clear. The second case is when the node has an indefinite edge to n' which is the failure reason. In this case n is the failure node since refining the value of the edge may change the value of n. The failure reason is then defined to be an atomic proposition with an indefinite value in the first case, and to an indefinite transition in the second case. Note that the algorithm is heuristic in the sense that it

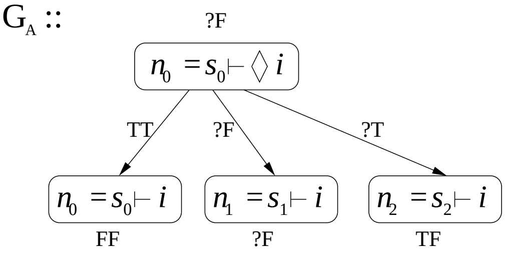


Figure 4.2: mc-graph for $\Diamond i$ on M_A

does not guarantee that all or any possible refinements of the failure node and reason will increase the information level of the result. Furthermore, it may even be the case that a refinement step decreases the information level of the result. The algorithm greedily searches for a failure node and reason which is most likely to increase the result with respect to the information order. On the other hand, the algorithm will not return as a failure node a node whose refinement will certainly not affect the result. For instance, subtrees in which the root node is definite are ignored, and their nodes will not be returned as failure node and reason.

Example 4.9 *Consider the mc-graph in Figure 4.2. This is a mc-graph created for verifying the property $\Diamond i$ on the abstract model M_A in Figure 4.1(b). For the node $n_0 = s_0 \vdash \Diamond i$ there are three possible failure nodes and reasons. The first is n_0 itself being the failure node and the edge to node n_2 being the reason. The second is node n_1 being the failure node and reason, and the third is node n_0 itself and the edge to n_1 being the reason.*
Assume that the first possibility has been chosen, namely, the node n_0 is the failure node and the edge to node n_2 is the reason. A refinement of M_A based on this choice results in the model M'_A (Figure 4.1(c)). Increasing the information level of the edge from s_0 to s_2 has been done here by splitting the state s_0 into two states.

Recall that refinement is done by splitting states. Thus, every abstract state of the model will represent less concrete states. If the underlying concrete model is finite, then there is only a finite number of possible such refinements. We conclude that there is a finite number of refinement steps possible. This is formalized in the following lemma.

Lemma 4.10 *If M_C is finite then in a finite number of refinement steps the model checking result will be the same as the one in the underlying concrete model, M_C.*

Proof: At each refinement step we split states. These means that every abstract state of the model will abstract less concrete states. Since the underlying concrete model is finite, then there is a finite number of splitting refinements available.

Thus, after a finite number of refinement steps, the value of all transitions and the value of every atomic proposition from the formula will be definite. By Theorem 3.11, the value of the formula on the refined model will be definite as well. Each refined model, M'_A is created such that it is an abstraction of the concrete model, M_C, thus $M_C \preceq M'_A$. By Theorem 4.3, for every closed L_μ formula φ, $\|\varphi\|^{M'_A} \leq_i \|\varphi\|^{M_C}$. We can then conclude that if the result on the abstract model is definite, then it is equal to the result on the concrete model.

□

Chapter 5

Partial Coloring and Subgraphs

In this chapter we investigate properties of the **mc-algorithm** (Algorithm 2.12). In particular, we present sufficient conditions under which a subgraph of a mc-graph can be evaluated “correctly” (the notion “correct” will be formally defined later) without considering the full mc-graph. The results presented in this chapter will assist us when presenting our compositional model checking framework in Chapter 6. There, we will construct a subgraph of the mc-graph of the composed system, based on the mc-graphs of the components. We will exploit the results from model checking the components in order to model check the subgraph of the composed system. Thus, we will avoid the construction of the full mc-graph for the composed system.

In the rest of the chapter, G denotes a multi-valued mc-graph over a multi-valued structure $\mathcal{B}= (L, \leq_t, \leq_i, \neg)$.

The purpose of the following definition is to identify those nodes in the mc-graph that can be ignored in the process of model checking, since they do not change the model checking result. These nodes will not be included in the subgraph to which we will apply model checking.

Definition 5.1 *Let G be a mc-graph and let $g : N \to L$ be a function. For a non-terminal node n in G, and two nodes n' and n'' which are sons of n, n'* covers n'' under *g with respect to n, if one of the following holds:*

- *n is a $\wedge$-node and*

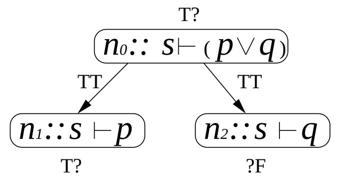


Figure 5.1: Influencing sons on mc-graph

1. *$(\neg R(n,n') \vee g(n')) \ \leq_t \ (\neg R(n,n'') \vee g(n''))$, and*
2. *For every $v', v'' \in L$: if $g(n') \leq_i v'$ and $g(n'') \leq_i v''$ then $(\neg R(n,n') \vee v') \ \leq_t \ (\neg R(n,n'') \vee v'')$.*

- *n is a $\vee$-node and*

1. *$(R(n,n'') \wedge g(n'')) \ \leq_t \ (R(n,n') \wedge g(n'))$, and*
2. *For every $v', v'' \in L$: if $g(n') \leq_i v'$ and $g(n'') \leq_i v''$ then $(R(n,n'') \wedge v'') \ \leq_t \ (R(n,n') \wedge v')$.*

Intuitively, a son n' covers a son n'' in the sense that if g defines the value of the sons, then it suffices to take into account n' (and ignore n'') in order to determine the value of the node n. In our setting, g will sometimes only provide a lower bound with respect to the information order on the value of the nodes. However, the second requirement ensures that the covering holds as well for every g' such that $\forall n \in N$: $g'(n) \geq_i g(n)$. Note that the notion of covering defines a partial order on the nodes of the mc-graph. As a result, for every covered node n'' there exists a covering node n' such that n' is non-covered.

Example 5.2 *Consider the mc-graph in Figure 5.1. Assume the underlying structure is the 3×3 structure (Figure 3.1(e), (f)). By the definition of the model, the value of the edges are $R(n_0,n_1) = R(n_0,n_2) = TT$. Let $g(n_1) = T?$ and $g(n_2) = ?F$. We show that n_2 is covered by n_1 under g. Clearly, requirement (1) holds. For requirement (2), we need to show that $\forall v_1, v_2 \in L$: if $g(n_1) \leq_i v_1$ and $g(n_2) \leq_i v_2$ then $(R(n_0,n_2) \wedge v_2) \leq_t (R(n_0,n_1) \wedge v_1)$. Specifically, we need to show that $\forall v_1 \in \{T?, TT, TF\}$, $\forall v_2 \in \{?F, FF, TF\}$: $TT \wedge v_2 \leq_t TT \wedge v_1$, which obviously holds.*

In the example, the value given to n_0 based on the sons n_1 and n_2 is the same as if only the son n_1 had been considered. We will next exploit this property.

Let G be a mc-graph and let $g : N \rightarrow L$ be a function. For a non-terminal node n in G, and a node n' which is a son of n, we say that n' is a *non-covered son of n under g* if the following holds. For every node $n'' \neq n'$ which is a son of n, n'' *does not* cover n' under g.

Definition 5.3 *Let G be a mc-graph and χ its mc-function. A subgraph G' of G is* closed *if every node n in G' is either terminal in G', or G' contains all non-covered sons of n under χ and corresponding edges.*

Definition 5.4 *Let G be a mc-graph, χ its mc-function, and $\chi_I : N \rightarrow L$ a partial mc-function. χ_I is* correct *with respect to χ if for every node n in G, if $\chi_I(n)$ is defined, then $\chi_I(n) = \chi(n)$.*

The **mc-algorithm** evaluates the nodes iteratively, starting from terminal nodes. As a result, if the algorithm is given a correct partial mc-function which is defined over (at least) all the terminal nodes, then it will extend the given valuation to the rest of the graph in a correct way.

Theorem 5.5 *Let G be a mc-graph and χ its mc-function. Consider a closed subgraph G' of G with a partial mc-function χ_I which is correct with respect to χ and defined over (at least) all the terminal nodes in G'. Then applying the* ***mc-algorithm*** *on G' with χ_I as an initial valuation (replacing val) results in a mc-function χ' of G' such that for every n in G', $\chi'(n) = \chi(n)$.*

Proof: We need to prove that when χ_I is used as the initial mc-function for the **mc-algorithm**, the evaluation of G' will be the same as the evaluation of G. For the terminal nodes of G' this is clear, since the initial value χ_I is defined over all terminal nodes and χ_I is correct with respect to χ for these nodes.

For a non-terminal node n in G': We show that the value of n in G depends only on the non-covered sons. Since all the non-covered sons of n in G are also in G', then the value of n in G' will be the same as its value in G.

Recall that $\vee$ and $\Diamond$ nodes are evaluated according to the following: $\chi(n) = \bigvee\{R(n,n') \wedge \chi(n') | R(n,n') \neq false\}$. Let n' and n'' be sons of n in G, such that n' covers n'' under χ with respect to n. By the definition of covering, this means that $\big(R(n,n'') \wedge \chi(n'')\big) \leq_t \big(R(n,n') \wedge \chi(n')\big)$. $a \leq_t b$ implies $a \vee b = b$. Thus, $\chi(n)$ will not be affected by the removal of the covered son n'', if there exists a son n' that covers n''. The covering defines a partial order, thus there always exists a covering son that is non-covered. By the definition of closed subgraph, all non-covered sons of n are included in G'. As a result, this is enough to update the value of n correctly.

Similarly, $\wedge$ and $\Box$ nodes are evaluated according to: $\chi(n) = \bigwedge\{\neg R(n,n') \vee \chi(n') | R(n,n') \neq false\}$. Again, this means that a valuation based on the non-covered sons will give the same result as a valuation based on all sons.

□

Chapter 6

Compositional Model Checking

In this chapter we define the composition of two models. We then present our framework for model checking a property on the composed system. The first step in the framework is presenting how one model can be "lifted" to be an abstraction of the composed system. The "lifted" model enables us to model check the required property on a single component, and deduce from that on the composed system. If nothing can be deduced from model checking each (lifted) component separately, the composed system should be considered. Having considered each component separately will help us in model checking the composed system, without fully constructing it. Based on the results of Chapter 5, we will be able to deduce about the full system from a partially constructed composed system.

6.1 Composition of Models

In compositional model checking the goal is to verify a formula φ on a composed system $M_1||...||M_n$. For simplicity, we will consider systems where $n = 2$, though our framework can easily be extended to any n. In our setting the composed system is $M_1||M_2$, where M_1 and M_2 are multi-valued (abstract) models defined over the same CPDB $\mathcal{P}(\mathcal{B}(\mathcal{D}))= (L, \leq_t, \leq_i, \neg)$. Thus, for $i \in \{1, 2\}$ there exists at least one concrete model M_i^c defined over $\mathcal{D}$, for which $M_i^c \preceq M_i$. We assume that if the model M_i is abstract (i.e., contains indefinite values), then it can be refined with respect to its underlying concrete model, M_i^c. Let $M_i = \langle \mathcal{B}, AP_i, S_i, s_0^i, R_i, \Theta_i \rangle$, we use $\bar{i}$ to denote the remaining index in $\{1, 2\} \setminus \{i\}$.

Definition 6.1 *Let $s_1 \in S_1$, $s_2 \in S_2$ be states in M_1 and M_2, respectively. Then, s_1, s_2 are* weakly composable *if for every $p \in AP_1 \cap AP_2 : \Theta_1(p)(s_1) \oplus \Theta_2(p)(s_2)$ is defined.*

Note that $\oplus$ might be undefined since $\mathcal{B}$ is a CPDB (Definition 3.7). Intuitively, if $\oplus$ is defined, then the composition of the states is consistent on all atomic propositions.

Definition 6.2 *States $s_1 \in S_1$, $s_2 \in S_2$ are* composable *if they are weakly composable, and for every $p \in AP_1 \cap AP_2 : \Theta_1(p)(s_1)$ and $\Theta_2(p)(s_2)$ are definite.*

In fact, since the definite values in CPDB are the highest in the information order, if s_1 and s_2 are composable, then for every $p \in AP_1 \cap AP_2$, $\Theta_1(p)(s_1) = \Theta_2(p)(s_2)$.

We say that M_1 and M_2 are *composable* if their initial states, s_0^1 and s_0^2 are composable.

Example 6.3 *Consider the two (abstract) models M_1 and M_2 in Figure 6.1(a). The underlying multi-valued structure is the 3×3 structure (described in Figure 3.1(e),(f)). The joint labelling of these models is $AP_1 \cap AP_2 = \{r\}$. The states s_0 and t_0 in M_1 and M_2, respectively, are composable. Similarly s_1, t_1 and s_2, t_2 are composable as well. However, states $s_3; t_3$, $s_2; t_3$, $s_1; t_3$ and $s_3; t_2$ are weakly composable (for example, for s_2 and t_3 $TT \oplus ?T$ is defined). Note that states s_1 and t_0, for example, are not weakly composable as $TT \oplus FF$ is not defined.*

In our setting M_1 and M_2 synchronize on the joint labelling of the states. Based on that, we now define composition of models.

Definition 6.4 *Let M_1 and M_2 be composable models. Their composition, denoted $M_1 || M_2$, is the multi-valued Kripke model $M = \langle \mathcal{B}, AP, S, s_0, R, \Theta \rangle$, where:*

- $AP = AP_1 \cup AP_2$,

- $S = \{(s_1, s_2) \in S_1 \times S_2 | \ s_1, \ s_2$ *are weakly composable*$\}$,
- $s_0 = (s_0^1, s_0^2)$,
- *For each* $(s_1, s_2), (t_1, t_2) \in S$:
  If t_1, t_2 *are composable, then* $R((s_1, s_2), (t_1, t_2)) = R_1(s_1, t_1) \wedge R_2(s_2, t_2)$. *Otherwise, if* t_1, t_2 *are weakly composable, then* $R((s_1, s_2), (t_1, t_2)) = R_1(s_1, t_1) \wedge R_2(s_2, t_2) \wedge \bot$.
- *For each* $(s_1, s_2) \in S$ *and* $p \in AP$:
  If $p \in AP_1 \cap AP_2$ *then* $\Theta(p)(s_1, s_2) = \Theta_1(p)(s_1) \oplus \Theta_2(p)(s_2)$.
  If $p \in AP_i \setminus AP_{\bar{i}}$ *then* $\Theta(p)(s_1, s_2) = \Theta_i(p)(s_i)$.

Composable states in the abstract model represent a composed state in the concrete composed model, for *every* concretization of the models. In contrast, for weakly composable but not composable states in the abstract model it is not guaranteed that these states represent a composed state in the concrete composed model.

Example 6.5 *Consider the models M_1 and M_2 in Figure 6.1(a). States s_0 and t_0 in M_1 and M_2, respectively, are composable. Indeed, by the definition of mixed simulation, since s_0 is the initial state, in* every *concretization of M_1 there exists a state s_0', which is abstracted by s_0. The value of r in s_0' is FF. Similarly, for every concretization of M_2 there exists a state t_0', which is abstracted by t_0, for which r is FF. Thus for* every *concretization of M_1 and M_2 there exists a concrete state (s_0', t_0') in the composed model. This state is represented by (s_0, t_0) in the composed abstract model. On the other hand, consider states s_3 and t_3 in M_1 and M_2, respectively. These states are weakly composable. Possible concretizations M_1' and M_2' of M_1 and M_2 are such that s_3' and t_3' which are concretizations of s_3 and t_3, assign the value TF and TT to r, respectively. Thus, (s_3',t_3') is not a state in $M_1'||M_2'$. Consequently, (s_3, t_3) does not represent any concrete state in $M_1'||M_2'$.*

The definition of the states in the composed model enables composition of states that are weakly composable but not composable. Such states do not exist in a composed concrete model (since the values of all atomic propositions in a concrete model are maximal with respect to the information order). However, they might exist when considering a composed *abstract* model. Unlike composable states, the weakly composable states in a composed abstract model may not have any corresponding state in the underlying concrete model. This is because in the concrete model, where the information level of some atomic propositions increases, the states might disagree on some p in their joint labelling.

Even though we are enabling weakly composable states which might not exist in the underlying concrete model, we want the abstract composed model to be an abstraction of the concrete composed model (i.e., we want to maintain a mixed simulation relation between these models). This is done with the definition of R. In the case where the target states are composable, the definition of R is immediate. If the target states are weakly composable but not composable, then we want to take into account the possibility that the transition does not exist. Defining its value to be $\perp$ achieves this goal. However, we can in fact guarantee more than that (in terms of the information order) by taking the meet with respect to truth order with $\perp$. This ensures that the value of the composed transition is not "more true" than $\perp$, but may be "more false" than $\perp$, and thus more informative. More precisely, consider the CPDB $\mathcal{P}(\mathcal{B}(\mathcal{D}))$ isomorphic to $\mathcal{B}$, where $\mathcal{D}= (D, \leq, \neg)$ is a De Morgan algebra. Then $\perp$ is defined as $\langle d_\perp, d_\perp \rangle \in D \times D$ where for every a in D, $d_\perp \leq a$. Thus, for every $\langle a, b \rangle \in D \times D$, $\langle a, b \rangle \wedge \perp = \langle d_\perp, b \rangle$, which means that the falsity level of $\langle a, b \rangle$ is preserved, whereas the truth level is minimal.

Allowing weakly composable states gives freedom to the user when abstracting each of the models, as all atomic propositions can be abstracted. In contrast, in [54], where composition of 3-valued models is discussed, joint labelling cannot be abstracted, thus all composable states in the abstract model represent composable states in the concrete model. There is a tradeoff presented with this freedom. On the one hand, the user can define a very coarse abstraction in each of the separate models. On the other hand, the abstract composed model might now include more states that do not represent any state in the concrete model.

Example 6.6 *Consider the two (abstract) components described in Figure 6.1(a). The atomic proposition o(output) is local to M_1, i(input) is local to M_2, and r(receive) is a joint atomic proposition that M_1 and M_2 synchronize on. The composition of these models is given in Figure 6.1(b). The dashed states in the composed model ((s_3, t_3), (s_2, t_3) and (s_3, t_2)) are states which are weakly composable but not composable, as in all of these states the atomic proposition $r \in AP_1 \cap AP_2$ is not definite in (at least) one* *of the composed states. Note, however, that in all of these states r has a definite value in the composed state. This refers to the fact that* if *there is a concrete state which is represented by, for example (s_2, t_3), then its value on r is TT. However, it is not guaranteed that there is such a concrete state. For instance, if s_3 is refined to a (concrete) state s'_3 in which $r = TF$ then s_3 cannot be composed with t_3 or any refinement of t_3.*

6.2 Lifting Models

Next, we define *lifting* of models for the purpose of compositional verification. The idea is to view each model M_i as a partial model that abstracts $M_1||M_2$.

From now on we fix AP to be $AP_1 \cup AP_2$.

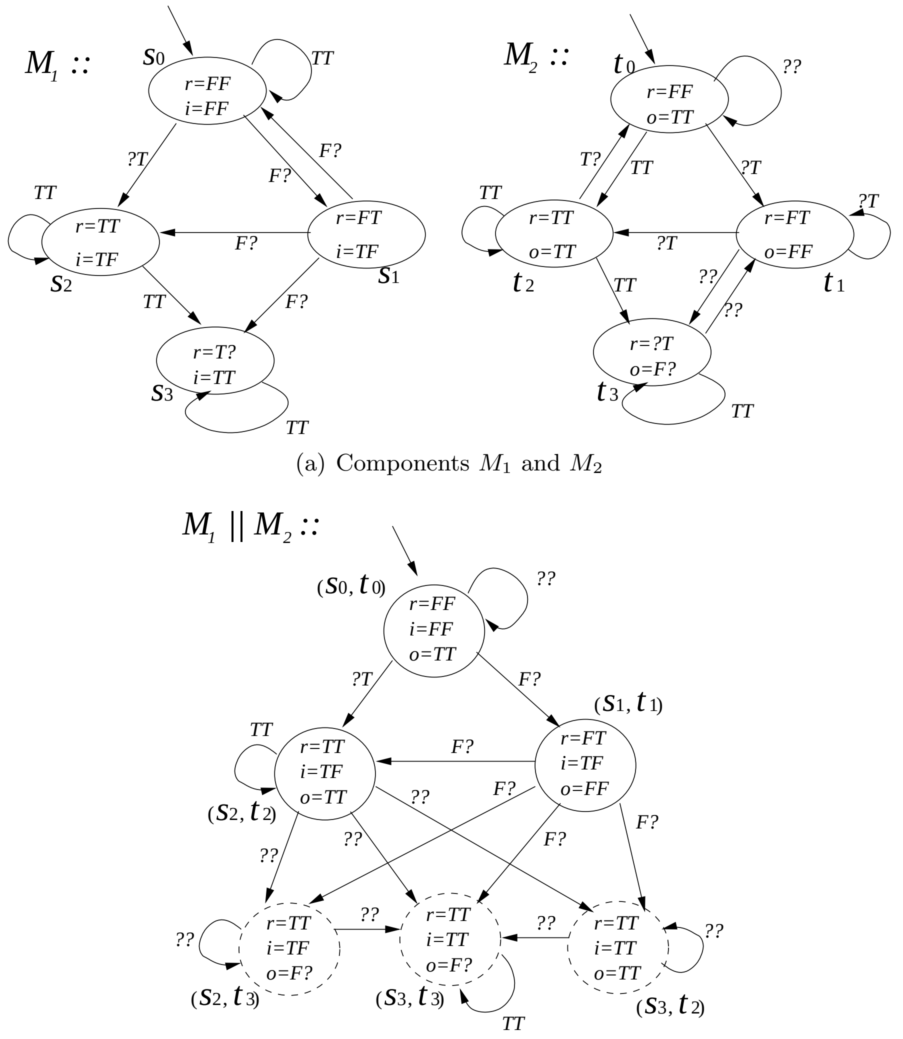
Definition 6.7 *The lifted model of $M_i = \langle\ \mathcal{B}, AP_i, S_i, s_0^i, R_i, \Theta_i\rangle$ is $M_i\!\uparrow = \langle\ \mathcal{B}, AP, S_i, s_0^i, R_i\!\uparrow, \Theta_i\!\uparrow\rangle$ where:*

- *For each $s_i, t_i \in S_i$: $R_i\!\uparrow(s_i, t_i) = R_i(s_i, t_i) \wedge \perp$.*
- *For each $s_i \in S_i$ and $p \in AP$:*
  If $p \in AP_i$ then $\Theta_i\!\uparrow(p)(s_i) = \Theta_i(p)(s_i)$.
  If $p \in AP \setminus AP_i$ then $\Theta_i\!\uparrow(p)(s_i) = \perp$.

The value of each literal over $AP \setminus AP_i$ in each state of $M_i\!\uparrow$ is minimal with respect to the information order ($\perp$). Indeed, its value in M will be determined by $M_{\bar{i}}$. In addition, each transition of M_i is also uncertain, in the sense that it cannot be "more true" than $\perp$. This is because in the composition it might be removed if a matching transition does not exist in $M_{\bar{i}}$.

Example 6.8 *The lifted models for the components in Figure 6.1(a) are presented in Figure 6.2. Observing $M_1\!\uparrow$, note that the atomic proposition o is now defined on all states, and is given the value ??. Note also that all transitions become uncertain. Similarly, the atomic proposition i is defined with a value ?? on all states of $M_2\!\uparrow$.*

Each of the lifted models is an abstraction of the composed system. This is formalized in the following theorem.



(a) Components M_1 and M_2

(b) The composed model $M_1||M_2$

Figure 6.1: Components M_1, M_2 and their composition

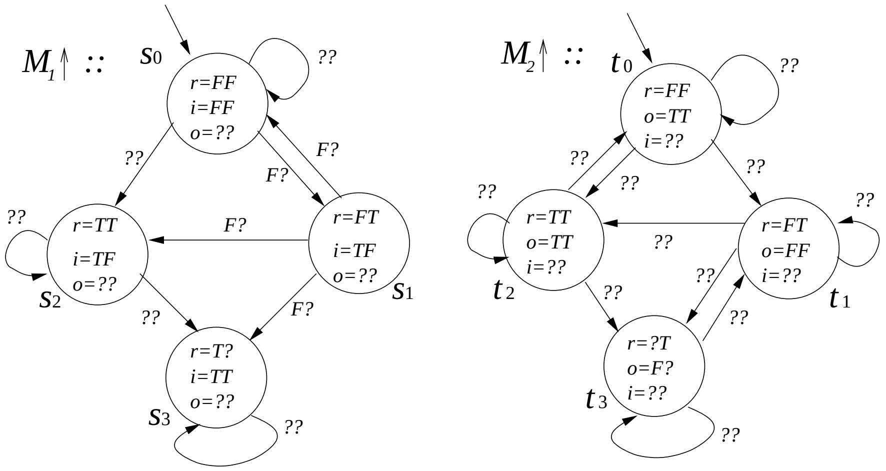


Figure 6.2: Lifted models for M_1 and M_2

Theorem 6.9 *$M_1||M_2 \preceq M_1\!\uparrow$. The mixed simulation relation $H \subseteq S \times S_1$ is given by $\{((s_1, s_2), s_1)|(s_1, s_2) \in S\}$. Similarly for M_2, $M_1||M_2 \preceq M_2\!\uparrow$.*

Proof: We prove that $M_1||M_2 \preceq M_1\uparrow$. The proof for M_2 is similar. Let $H \subseteq S \times S_1$ be the relation defined by $\{((s_1, s_2), s_1)|(s_1, s_2) \in S\}$. That is, a state, $s_1 \in M_1\uparrow$ is related by H to all the states of $M_1||M_2$ where the corresponding state in the pair is s_1.

First note that the initial state is in H, since by definition of the composition, the initial state of $M_1||M_2$ consists of the initial states of both components.

Let $(s, s_1) \in H$. Then:

1. Prove that $\forall p \in AP : \Theta_1\!\uparrow\!(p)(s_1) \leq_i \Theta(p)(s)$
   - $p \in AP_1 \setminus AP_2$: By the definition of the composition: $\Theta(p)(s) = \Theta_1(p)(s_1)$. By the definition of $M_1\uparrow$, $\Theta_1\uparrow(p)(s_1) = \Theta_1(p)(s_1)$. Thus, $\Theta(p)(s) = \Theta_1\uparrow(p)(s_1)$, which immediately implies $\Theta_1\uparrow(p)(s_1) \leq_i \Theta(p)(s)$.
   - $p \in AP_1 \cap AP_2$: By the definition of the composition: $\Theta(p)(s) = \Theta_1(p)(s_1) \oplus \Theta_2(p)(s_2)$. Since $a \leq_i a \oplus b$ then $\Theta_1(p)(s_i) \leq_i$

  $\Theta_1(p)(s_1) \oplus \Theta_2(p)(s_2)$. By definition of $M_1 \uparrow$, $\Theta_1 \uparrow (p)(s_1) = \Theta_1(p)(s_1)$. Thus, $\Theta_1 \uparrow (p)(s_1) \leq_i \Theta(p)(s)$.

  - $p \in AP_2 \setminus AP_1$: By the definition of the composition: $\Theta(p)(s) = \Theta_2(p)(s_2)$. By the definition of $M_1 \uparrow$: $\Theta_1 \uparrow (p)(s_1) = \bot$. Thus, $\Theta_1 \uparrow (p)(s_1) \leq_i \Theta(p)(s)$.

2. Prove that for every $t \in S$ such that $R(s,t) \neq false$ there exists $t_1 \in S_1$ such that $(t, t_1) \in H$ and $R_1 \uparrow (s_1, t_1) \leq_i R(s,t)$.
   Let $t = (t_1, t_2) \in S$ and let $R(s,t) \neq false$.
   Since $R(s,t) \neq false$, we have $R_1(s_1,t_1) \neq false$. By the definition of $M_1 \uparrow$, $R_1 \uparrow (s_1,t_1) = R_1(s_1,t_1) \wedge \bot$. By definition of the composed model, $R(s,t) = R_1(s_1,t_1) \wedge R_2(s_2,t_2)$ (if t_1 and t_2 are composable) or $R_1(s_1,t_1) \wedge R_2(s_2,t_2) \wedge \bot$ (if t_1 and t_2 are weakly composable and not composable). Since $\bot \leq_i R_2(s_2,t_2)$, and since $\wedge$ is monotone with respect to $\leq_i$, we conclude that $R_1 \uparrow (s_1,t_1) \leq_i R(s,t)$. For t_1, $(t,t_1) \in H$ and $R_1 \uparrow (s_1,t_1) \leq_i R(s,t)$. Thus t_1 satisfies the proof requirement.
3. Prove that for every $t_1 \in S_1$ such that $R_1 \uparrow (s_1,t_1) \not\leq_i false$ there exists $t \in S$ such that $(t,t_1) \in H$ and $R_1 \uparrow (s_1,t_1) \leq_i R(s,t)$.
   Let $t_1 \in S_1$ and $R_1 \uparrow (s_1,t_1) \not\leq_i false$.
   By definition of $M_1 \uparrow$, the value of $R_1 \uparrow (s_1,t_1) = R_1(s_1,t_1) \wedge \bot$. $\bot \leq_i false$ and $R_1(s_1,t_1) \leq_i R_1(s_1,t_1)$. Since $\wedge$ is monotone with respect to $\leq_i$, we conclude that $R_1 \uparrow (s_1,t_1) = R_1(s_1,t_1) \wedge \bot \leq_i R_1(s_1,t_1) \wedge false = false$. Thus, the proof requirement holds trivially, there is no t_1 such that $R_1 \uparrow (s_1,t_1) \not\leq_i false$ in $M_1 \uparrow$.

□

Example 6.10 *Based on Theorem 6.9, we conclude that the lifted model $M_1 \uparrow$ (Figure 6.2) is an abstraction of the composed system $M_1||M_2$ (Figure 6.1(b)), that is, $M_1||M_2 \preceq M_1 \uparrow$. The mixed simulation is given by: $H_1 = \{((s_0,t_0),s_0),((s_1,t_1),s_1),((s_2,t_2),s_2),((s_2,t_3),s_2),((s_3,t_2),s_3),((s_3,t_3),s_3)\}$. Similarly, $M_1||M_2 \preceq M_2 \uparrow$, and the mixed simulation is given by: $H_2 = \{((s_0,t_0),t_0),((s_1,t_1),t_1),((s_2,t_2),t_2),((s_3,t_2),t_2),((s_3,t_3),t_3),((s_3,t_3),t_3)\}$.*

Since each $M_i \uparrow$ abstracts $M_1||M_2$, we are able to first consider each component separately: Theorem 4.3 ensures that for every closed L_μ formula φ, $\|\varphi\|^{M_i\uparrow} \leq_i \|\varphi\|^{M_1||M_2}$. In particular, $\|\varphi\|^{M_1\uparrow} \oplus \|\varphi\|^{M_2\uparrow}$ is defined, and a definite result on one of the components suffices to determine a definite value on $M_1||M_2$. Note that a definite value on $M_1||M_2$ can be achieved even if both $\|\varphi\|^{M_i\uparrow}$ are indefinite, but $\|\varphi\|^{M_1\uparrow} \oplus \|\varphi\|^{M_2\uparrow}$ is definite.

Example 6.11 *Consider the formula $\varphi = i \vee o$, the lifted models $M_1\uparrow$ and $M_2\uparrow$ (Figure 6.2), and the composed model $M_1||M_2$ (Figure 6.1(b)). $\|\varphi\|^{M_1\uparrow}(s_2) = T?$ and $\|\varphi\|^{M_2\uparrow}(t_2) = ?T$, thus the evaluation of φ on each of the lifted models results in an indefinite answer. Since $M_1||M_2 \preceq M_1\uparrow$ we can conclude $\|\varphi\|^{M_1\uparrow}(s_2) \leq_i \|\varphi\|^{M_1||M_2}((s_2,t_2))$, thus $T? \leq_i \|\varphi\|^{M_1||M_2}((s_2,t_2))$. Similarly, $\|\varphi\|^{M_2\uparrow}(t_2) = ?T \leq_i \|\varphi\|^{M_1||M_2}((s_2,t_2))$.*
Since $\|\varphi\|^{M_1\uparrow}(s_2) \oplus \|\varphi\|^{M_2\uparrow}(t_2) \leq_i \|\varphi\|^{M_1||M_2}((s_2,t_2))$, we conclude: $T? \oplus ?T = TT \leq_i \|\varphi\|^{M_1||M_2}((s_2,t_2))$. Thus, $\|\varphi\|^{M_1||M_2}((s_2,t_2)) = TT$, since TT is maximal by the information order.

A more typical case is when the valuation of φ on both $M_1\uparrow$ and $M_2\uparrow$ is indefinite. This reflects the fact that φ depends on both components and on their synchronization. Typically, such a result requires some refinement of the abstract model. Considering the composition of the two components is a refinement of the lifted models. Still, having considered each component separately can guide us into focusing on the places where we indeed need to consider the composition of the components. Thus, we avoid the construction of the full composed model.

6.3 Building The Product Subgraph

In this section we use the mc-graphs of $M_1\uparrow$ and $M_2\uparrow$ for building a subgraph for $M_1||M_2$, and by that avoid building the full composed model. The mc-graphs of the two components present all the information gained from model checking each component. To resolve the indefinite result, we first try to compose the parts of the mc-graphs which might be needed to determine the value of the formula.

Definition 6.12 *For every $i \in \{1,2\}$, let $G_i = (n_i^0, N_i, E_i)$ be the mc-graph of $M_i\uparrow$, with χ_i its mc-function. $\chi_f : N_1 \times N_2 \to L$ is a function defined by $\chi_f(n_1,n_2) = \chi_1(n_1) \oplus \chi_2(n_2)$.*
$E_f : (N_1 \times N_2) \times (N_1 \times N_2) \to L$ is a function defined as follows. Let $n_i' = (s_i' \vdash \varphi') \in N_i$, then $E_f((n_1,n_2),(n_1',n_2')) = R_1(s_1,s_1') \wedge R_2(s_2,s_2')$ if s_1' and

s'_2 are composable, and $E_f((n_1, n_2), (n'_1, n'_2)) = R_1(s_1, s'_1) \wedge R_2(s_2, s'_2) \wedge \bot$ if s'_1 and s'_2 are weakly composable but not composable.

Let $G = (n^0, N, E)$ be a mc-graph and let $f : N \to L$ and $e : N \times N \to L$ be two functions. For a non-terminal node n in G, and two nodes n' and n'' which are sons of n, we abuse the notion of covering (Definition 5.1) and say that n' covers n'' under f *and* e with respect to n, if n' covers n'' under f with respect to n when e replaces the transition relation R.

Definition 6.13 ***(Product Graph)*** *Let M_1 and M_2 be two composable models. For every $i \in \{1, 2\}$, let $G_i = (n^0_i, N_i, E_i)$ be the mc-graph of $M_i\uparrow$, with an initial node $n^0_i = (s^0_i \vdash \varphi) \in N_i$. Also let χ_i be the mc-function of G_i. The product graph of G_1 and G_2, denoted $G_\| = (n^0_\|, N_\|, E_\|)$, is defined as the least graph that obeys the following:*

- *$n^0_\| = (s^0_1, s^0_2) \vdash \varphi$ is the initial node in $G_\|$.*
- *Let $n_1 = s_1 \vdash \psi$, $n_2 = s_2 \vdash \psi$ be such that $(n_1, n_2) \in N_\|$, and $\chi_f(n_1, n_2)$ is indefinite. Then for every $n'_1 = (s'_1 \vdash \psi') \in N_1$ and $n'_2 = (s'_2 \vdash \psi') \in N_2$, if the following holds:*

  1. *s'_1, s'_2 are weakly composable, and*
  2. *$E_1(n_1, n'_1) \neq false$ and $E_2(n_2, n'_2) \neq false$, and*
  3. *(n'_1, n'_2) is not covered under χ_f and E_f with respect to (n_1, n_2).*

  Then:

  1. *$(n'_1, n'_2) \in N_\|$, and*
  2. *$E_\|((n_1, n_2), (n'_1, n'_2)) = E_f((n_1, n_2), (n'_1, n'_2))$.*

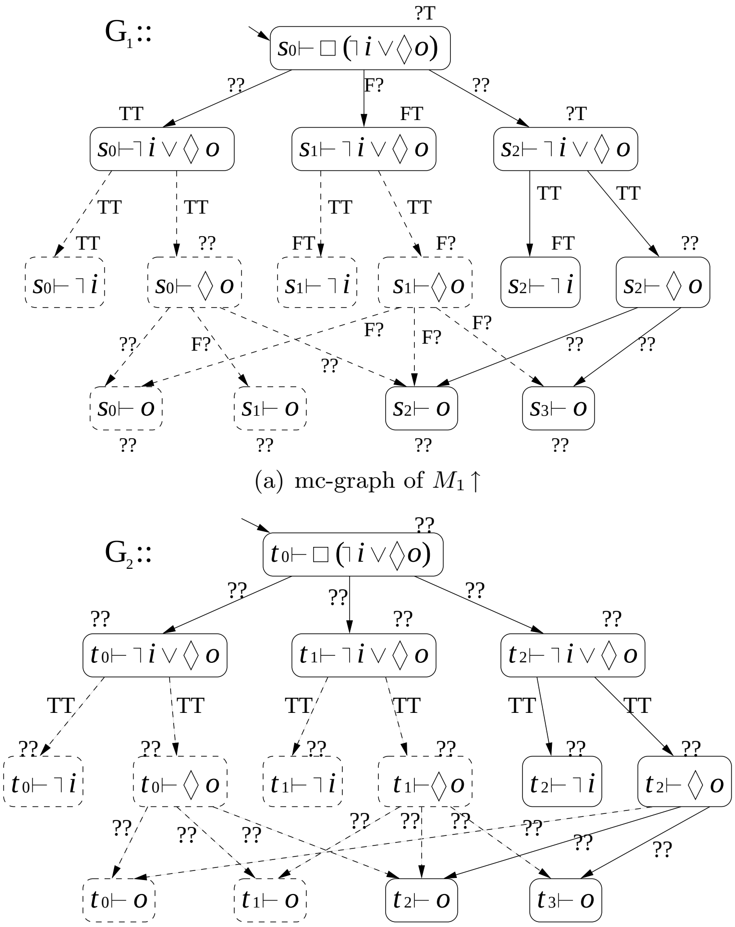
Lemma 6.14 *Let $G_\|$ be the product graph defined above. For every node $n \in N_\|$, $\chi_f(n)$ is defined.*

Proof: Recall that the $\oplus$ operation might be undefined on CPDBs. Assume $n = (s_1, s_2) \vdash \psi$. χ_1 is a correct mc-function of G_1, thus $\|\psi^*\|^{M_1\uparrow} = \chi_1(s_1 \vdash \psi)$. ψ^* is a closed L_μ formula, then since $M_1||M_2 \preceq M_1 \uparrow$ we conclude that $\chi_1(s_1 \vdash \psi) \leq_i \|\psi^*\|^{M_1||M_2}(s_1, s_2)$. Similarly, $\chi_2(s_2 \vdash \psi) \leq_i \|\psi^*\|^{M_1||M_2}(s_1, s_2)$. As a result, $\chi_1(s_1 \vdash \psi) \oplus \chi_2(s_2 \vdash \psi) \leq_i \|\psi^*\|^{M_1||M_2}(s_1, s_2)$, which means that the $\oplus$ operation is defined for n. □

Note that the value of the edges in $G_\|$ is identical to their value in the composed model. This is because the product graph already refers to the complete system $M_1||M_2$. In contrast, the values of the edges in the mc-graphs of each component are all smaller or equal by the truth order than $\perp$.

The product graph is constructed by a top-down traversal on the mc-graphs of the two models, where, starting from the initial node, nodes that share the same formulas and whose states are weakly composable, will be considered. Whenever two non-terminal nodes n_1, n_2 are composed, if $\chi_f(n_1, n_2)$ is indefinite, then the outgoing edges are computed as the product of their outgoing edges, restricted to weakly composable nodes. In particular, this means that if a node in one mc-graph has no matching node in the other, then it will be omitted from the product graph. After computing all legal sons based on the outgoing edges, the nodes which are covered under χ_f will be removed, leaving as outgoing edges and nodes only nodes which are not covered under χ_f. In particular, when a terminal node of one mc-graph is composed with a non-terminal node of the other, the resulting node is a terminal node in $G_{||}$. Note that we compute χ_f and E_f only by need. Note also that whenever a definite node is composed with another node (definite or not), χ_f of the resulting node is definite, which makes it a terminal node in the product graph.

Example 6.15 *We wish to verify the property $\Box(\neg i \vee \Diamond o)$, which states that in all the successor states of the initial state, an input signal implies that there is a successor state where the output signal holds. The mc-graphs of the lifted models $M_1\uparrow$ and $M_2\uparrow$ are described in Figure 6.3(a) and Figure 6.3(b) respectively. The model checking on each of the models does not result in a definite answer, and we need to consider their composition. The parts that are actually composed are marked with solid lines. The product graph is shown in Figure 6.4. The edges get their actual value (based on E_f). The value of χ_f of each node is given on the node (un-parenthesized value). The nodes which are covered are marked with dashed lines. These nodes are created and removed on-the-fly, since they are covered, and their successors are not considered. The actual nodes that are included in the product graph are marked with solid lines.*
Note that the product graph considers only a small part of the compound system, as it takes advantage of the information from the separate components.



(a) mc-graph of $M_1\uparrow$

(b) mc-graph of $M_2\uparrow$

Figure 6.3: mc-graphs of $M_1\uparrow$ and $M_2\uparrow$. Solid lines mark composed subgraph

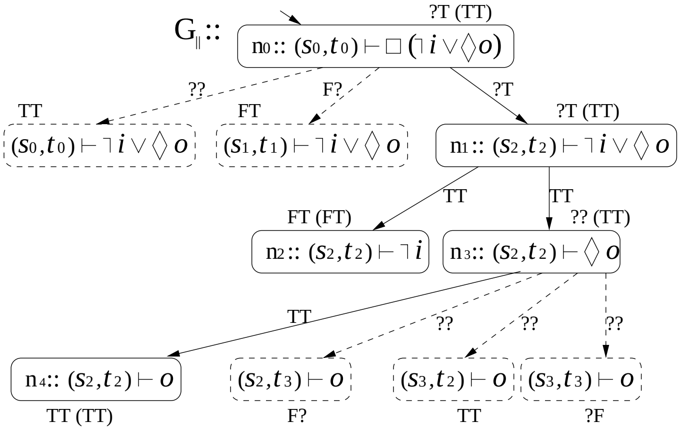


Figure 6.4: The product graph. Dashed nodes are covered. Solid lines mark actual product graph.

We accompany $G_{||}$ with an initial mc-function, χ_I, for its terminal nodes, based on the mc-functions of the two mc-graphs. We use the following observation:

Let $n = (s_1, s_2) \vdash \psi$ be a terminal node in $G_{||}$. Then at least one of the following holds. Either (a) at least one of $s_1 \vdash \psi$ and $s_2 \vdash \psi$ is a terminal node in its mc-graph; Or (b) $\chi_f(s_1 \vdash \psi, s_2 \vdash \psi)$ is definite; Or (c) both $s_1 \vdash \psi$ and $s_2 \vdash \psi$ are non-terminal but no outgoing edges were left in their composition.

Definition 6.16 *The initial mc-function χ_I of $G_{||}$ is defined as follows. Let $n = (s_1, s_2) \vdash \psi \in N_{||}$ be a terminal node. If it fulfills case (a) or (b), then $\chi_I(n) = \chi_1(s_1 \vdash \psi) \oplus \chi_2(s_2 \vdash \psi)$. If it fulfills case (c), then $\chi_I(n) = true$ if n is a $\wedge$-node, and $\chi_I(n) = false$ if n is a $\vee$-node. χ_I is undefined for the rest of the nodes.*

Theorem 6.17 *Let G be the mc-graph over $M_1||M_2$ with a mc-function χ. The resulting product graph $G_{||}$ is a closed subgraph of G. In addition, χ_I is defined over all the terminal nodes of $G_{||}$, and is correct with respect to χ.*

For the proof of Theorem 6.17 we need the following Lemmas.

Lemma 6.18 *Let $H \subseteq S_1 \times S_2$ be a mixed simulation relation from M_1 to M_2 ($M_1 \preceq M_2$). Let $G_1(M_1, \varphi_0)$ and $G_2(M_2, \varphi_0)$ be mc-graphs with their mc-functions, χ_1 and χ_2, which are semantically correct. Let $\psi \in Sub(\varphi_0)$ be a L_μ formula (not necessarily closed). Let $(s_1, s_2) \in H$ and let $s_1 \vdash \psi$, $s_2 \vdash \psi$ be nodes in G_1 and G_2,respectively. Then $\chi_2(s_2 \vdash \psi) \leq_i \chi_1(s_1 \vdash \psi)$.*

Proof: χ_1 is a semantically correct mc-function for G_1, thus $\|\psi^*\|^{M_1}(s_1) = \chi_1(s_1 \vdash \psi)$. Similarly, $\|\psi^*\|^{M_2}(s_2) = \chi_2(s_2 \vdash \psi)$. By definition, ψ^* is a closed L_μ, and therefore by Theorem 4.3, $\|\psi^*\|^{M_2}(s_2) \leq_i \|\psi^*\|^{M_1}(s_1)$. Thus, $\chi_2(s_2 \vdash \psi) \leq_i \chi_1(s_1 \vdash \psi)$ □

Lemma 6.19 *The initial coloring function χ_I of $G_{||}$ is defined for all terminal nodes, and is well defined.*

Proof: Recall the following observation. Let $n = (s_1, s_2) \vdash \psi$ be a terminal node in $G_{||}$. Then at least one of the following holds. Either (a) at least one of $s_1 \vdash \psi$ and $s_2 \vdash \psi$ is a terminal node in its mc-graph; Or (b) $\chi_f(s_1 \vdash \psi, s_2 \vdash \psi)$ is definite; Or (c) both $s_1 \vdash \psi$ and $s_2 \vdash \psi$ are non-terminal but no outgoing edges were left in their composition.

Based on the observation and by Definition 6.16, the initial coloring is defined for all terminal nodes. A terminal node can comply to more than one case of the observation. Thus, we need to show that the definition of χ_I is well defined. Specifically, this means that if a node $n = (s_1, s_2) \vdash \psi$ complies to both cases (b) and (c), then we require the definition of $\chi_I(n)$ for these cases to be equal.
$M_1||M_2 \preceq M_i \uparrow$ for $i \in \{1, 2\}$. Thus $\chi_i(s_i \vdash \psi) \leq_i \chi((s_1, s_2) \vdash \psi)$ (Lemma 6.18). This means that $\chi_1(s_1 \vdash \psi) \oplus \chi_2(s_2 \vdash \psi) \leq_i \chi((s_1, s_2) \vdash \psi)$. Since n complies with case (b), we know that $\chi_f(s_1 \vdash \psi, s_2 \vdash \psi) = \chi_1(s_1 \vdash \psi) \oplus \chi_2(s_2 \vdash \psi)$ is definite, thus $\chi_1(s_1 \vdash \psi) \oplus \chi_2(s_2 \vdash \psi) = \chi((s_1, s_2) \vdash \psi)$. n also complies with case (c), thus no outgoing edges were left in their composition. This means that in $M_1||M_2$, the state (s_1, s_2) has no outgoing edges. If n is a $\wedge$-node, then $\chi(n) = true$, and if n is a $\vee$-node, then $\chi(n) = false$. We can conclude that both definitions result in the same initial valuation, and thus the initial valuation is well defined for all initial nodes. □

We now return to the proof of Theorem 6.17.

Proof: We first show that $G_{||}$ is a closed subgraph of G. It is easy to see that it is a subgraph of G, since the structure in terms of the subformulas and edges is maintained. We now show that the subgraph is closed. Assume by way of contradiction that $G_{||}$ contains a non-terminal node n whose non-covered sons are not all included. Let n'' be such a non-covered son. Thus, n'' is a non-covered son in G. The nodes that correspond to the non-covered son n'' are included in the mc-graphs of the two components. Thus, n'' was removed during the building of the product graph. This can be done only if n'' is covered under χ_f. As n'' is covered under χ_f, there exists a node n', which covers n'' under χ_f. For every $n = (n_1, n_2)$, $\chi_f(n)$ is defined as $\chi_1(n_1) \oplus \chi_2(n_2)$. Due to the correctness of the valuation with respect to the multi-valued semantics, and by the mixed simulation relation, $\chi_1(n_1) \oplus \chi_2(n_2) \leq_i \chi(n)$, thus $\chi_f(n) \leq_i \chi(n)$. This relation holds for every node in N, thus $\chi_f(n') \leq_i \chi(n')$ and $\chi_f(n'') \leq_i \chi(n'')$. The value of the edges is the same in both $G_{||}$ and G. Then according to the definition of a covered son (Definition 5.1), n'' is covered by n' under χ in G as well. This contradicts the assumption that n'' is a non-covered son in G. We can conclude that $G_{||}$ is a closed subgraph of G.

It remains to be proven that χ_I is defined on all terminal nodes of $G_{||}$, and $\chi_I(n) = \chi(n)$ for every terminal node. By Lemma 6.19, χ_I is defined for all terminal nodes. We now prove that for every terminal node $n = (s_1, s_2) \vdash \psi \in N_{||}$: $\chi_I(n) = \chi(n)$. We prove this for every case of terminal node in $N_{||}$, based on the observation.

- At least one of $s_1 \vdash \psi$ and $s_2 \vdash \psi$ is a terminal node in its mc-graph. Assume without loss of generality that $n_1 = s_1 \vdash \psi$ is a terminal node in G_1. Then n_1 is of the form $s_1 \vdash p$ or $s_1 \vdash \neg p$ for $p \in AP$, or n_1 is of the form $s_1 \vdash \Diamond\varphi$ or $s_1 \vdash \Box\varphi$ where there are no transitions from s_1 in M_1. We show that $\chi_I(n) = \chi(n)$ based on the type of n_1:
  - If $n_1 = s_1 \vdash p$ for $p \in AP$. By the definition of the composed model $\Theta(p)((s_1, s_2)) = \Theta_1(p)(s_1) \oplus \Theta_2(p)(s_2)$. By correctness of the model checking algorithm, $\chi_i(n_i) = \Theta_i(p)(s_i)$ and $\chi(n) = \Theta(p)((s_1, s_2))$. χ_I is defined such that $\chi_I(n) = \chi_1(n_1) \oplus \chi_2(n_2)$, and we can then conclude that $\chi_I(n) = \chi(n)$.
  - The case where $n_1 = s_1 \vdash \neg p$ for $p \in AP$ is dual.
  - If $n_1 = s_1 \vdash \Diamond\varphi$, then by definition of the mc-graph, $\chi_1(n_1) =$

$false$. Thus, $\chi_I(n) = \chi_1(n_1) \oplus \chi_2(n_2) = false$.
Since there are no transitions from s_1 in M_1, then there are no transitions from (s_1, s_2) in $M_1||M_2$. As a result, by definition of the model checking algorithm, $\chi(n) = false$. We can then conclude that $\chi_I(n) = \chi(n)$.

  - The case where $n_1 = s_1 \vdash \Box\varphi$ is dual.

- $\chi_f(s_1 \vdash \psi, s_2 \vdash \psi)$ is definite. By definition, $\chi_f(n_1, n_2) = \chi_1(n_1) \oplus \chi_2(n_2)$. By correctness of the mixed simulation and correctness of the mc-functions $\chi_i(n_i) \leq_i \chi(n)$ (Lemma 6.18). Thus $\chi_1(n_1) \oplus \chi_2(n_2) \leq_i \chi(n)$. If $\chi_1(n_1) \oplus \chi_2(n_2)$ is definite, we can conclude that $\chi_1(n_1) \oplus \chi_2(n_2) = \chi(n)$, thus $\chi_I(n) = \chi(n)$.
- Both $s_1 \vdash \psi$ and $s_2 \vdash \psi$ are non-terminal nodes but no outgoing edges were left in their composition. Since no outgoing edges were left in their composition, we can conclude that there are no outgoing edges from (s_1, s_2) in $M_1||M_2$. Thus, $\chi(n) = false(true)$ if n is a $\vee$-node($\wedge$-node), and this is the color given to n in χ_I. Thus, $\chi_I(n) = \chi(n)$.

□

6.4 Compositional Model Checking Framework

Theorems 5.5 and 6.17 imply that applying the **mc-algorithm** on $G_{||}$ with χ_I results in a correct mc-function χ with respect to $G_{||}$. Thus, $\chi(n^0_{||})$ is the value of model checking φ on $M_1||M_2$. As a result, to model check φ on $M_1||M_2$ it remains to evaluate $G_{||}$. Note that the full graph for $M_1||M_2$ is not constructed.

Example 6.20 *Consider the product graph in Figure 6.4. By Theorem 6.17, this is a closed subgraph of the mc-graph over $M_1||M_2$ with mc-function χ. By Definition 6.16, the initial mc-function χ_I of $G_{||}$ is: $\chi_I(n_4) = TT$ and $\chi_I(n_2) = FT$. Note that these are the only terminal nodes in $G_{||}$.*
*Since this is a closed subgraph, then by Theorem 5.5, the **mc-algorithm** can be applied on it. The mc-function of $G_{||}$ is described in Figure 6.4, as the parenthesized value of each node in the subgraph. The value of the top node is definite (TT), and we can thus conclude that $\|\Box(\neg i \vee \Diamond o)\|^{M_1||M_2} = TT$.*

If the model checking result is indefinite (which is only possible if $M_1||M_2$ is abstract), then refinement is performed on each component separately, based on the failure node and reason returned. This is described in the following steps, which summarize our compositional algorithm for checking an alternation-free L_μ formula φ on $M_1||M_2$.

Step 1: Model check each $M_i\!\uparrow$ separately (for $i \in \{1, 2\}$):

1. Construct the mc-graph G_i for φ and $M_i\!\uparrow$.
2. Apply multi-valued model checking on G_i. Let χ_i be the resulting mc-function.

If $\chi_1(n_1^0)$ or $\chi_2(n_2^0)$ is definite, return the corresponding model checking result for $M_1||M_2$.

Step 2: Consider the composition $M_1||M_2$:

1. Construct the product graph $G_{||}$ of the mc-graphs G_1 and G_2.
2. Apply multi-valued model checking on $G_{||}$ (with the initial mc-function).

If $\chi_{||}(n_{||}^0)$ is definite, return the corresponding model checking result for $M_1||M_2$.

Step 3: Refine: Consider the failure node and reason returned by model checking $G_{||}$ (where $\chi_{||}(n_{||}^0)$ is indefinite).
If it is p for some $p \in AP_i$, then refine G_i;
Else let it be an edge $((s_1, s_2), (s'_1, s'_2))$. Then:

1. If s'_1 and s'_2 are weakly composable but not composable, refine both G_1 and G_2 according to $AP_1 \cap AP_2$.
2. If $R_i(s_i, s'_i) \leq_t R_{\bar{i}}(s_{\bar{i}}, s'_{\bar{i}})$, refine the edge $R_i(s_i, s'_i)$ in G_i.
3. If $R_i(s_i, s'_i)$ and $R_{\bar{i}}(s_{\bar{i}}, s'_{\bar{i}})$ are uncomparable, refine the mc-graph(s) in which the edge is indefinite.

Go to Step 1(2) with the refined mc-graphs.

Theorem 6.21 *For finite components, the compositional algorithm is guaranteed to terminate with a definite answer.*

Proof: Recall our assumption that every (abstract) model M_i has an underlying concrete model M_i^c. Based on the correctness of the failure node and reason finding algorithm, if the answer is not definite, then the algorithm returns a failure node and reason. Based on Lemma 4.10, for finite components, in a finite number of refinement steps the model checking result will be the same as the underlying concrete model, and thus will be definite, and the compositional algorithm will terminate. □

Chapter 7

Handling Full Distributive Bilattices

In the previous chapters we have presented the compositional framework for abstract and concrete models which were defined over CPDBs. In this chapter we discuss how our framework can be used for multi-valued structures that include inconsistent elements, and are described as full distributive bilattices.

Models which are defined over full bilattices include either transitions with inconsistent values, or inconsistent values of atomic propositions in some states (or both). Two examples where model checking is done based on models defined over full bilattices are STE ([50]) and YASM ([36]). In both the multi-valued structure used is the Belnap structure (Figure 3.1(a) and (b)). We will present our compositional framework in light of these examples.

In the rest of the chapter we first describe mixed simulation and refinement of multi-valued models over bilattices. We then give some background on STE and YASM, and continue to present how our compositional framework can be used for full bilattices, exemplifying on the above systems.

7.1 Mixed Simulation and Refinement of Multi-Valued Models Over Bilattices

In Chapter 4 we argued that a concrete model, M_c, defined over a De Morgan algebra $\mathcal{D}$ can be interpreted as a model over the CPDB $\mathcal{P}(\mathcal{B}(\mathcal{D}))$. Similarly, M_c can also be interpreted over the bilattice $\mathcal{B}(\mathcal{D})$= $(D \times D, \leq_i, \leq_t, \neg)$, where the values are all definite and consistent. We also argued that the relation between M_c and its abstraction, M_A can be described using the mixed simulation relation (Definition 4.1), where $M_c \preceq M_A$.

Assume now that the abstract model M_A is defined over the bilattice $\mathcal{B}(\mathcal{D})$, and it includes inconsistent values. In this case, the relation between the concrete model M_c and the abstract model M_A cannot be described using mixed simulation relation. This is because there are values (of either transitions or atomic propositions in some state) which are not an abstraction of values in the concrete model. Intuitively, the abstract model M_A is "abstract" in the sense that it does not define a concrete model. However, it is *not* an abstraction of any concrete model M_c, as it cannot be stated that M_A contains less information than M_c.

Resulting from the above observation, we will not discuss the connection between a concrete model M_c, defined over a De Morgan algebra $\mathcal{D}$, and an abstract model M_A, defined over $\mathcal{B}(\mathcal{D})$. The connection between these models depends on the way the abstract model is created.

When discussing models defined over a bilattice $\mathcal{B}(\mathcal{D})$, we will use the mixed simulation relation to describe the connection between two abstract models both defined over $\mathcal{B}(\mathcal{D})$. Both the definition of mixed simulation (Definition 4.1) and its connection to model checking (Theorem 4.3) remain the same for models defined over a bilattice $\mathcal{B}(\mathcal{D})$. Similarly, the definition of a closed subgraph (Definition 5.3) and the correctness of the **mc-algorithm** with respect to a closed subgraph (Theorem 5.5) remain the same for models defined over a bilattice $\mathcal{B}(\mathcal{D})$.

What is a refinement of a model defined over a bilattice $\mathcal{B}(\mathcal{D})$? As discussed in Chapter 4, the goal in refinement is to increase the information level of the result. We also argued that, with regards to CPDBs, definite values include "all the information possible", and thus should not be refined. When using distributive bilattices, definite values include values that are both consistent and inconsistent. Also, there might be two different elements, both definite, where one is higher in the information order than the other. We claim that as long as the value is definite, we do not need to refine it. Thus, a *true* or *false* result (which are definite values, but are not highest in the information order) are not values we wish to refine. Similar to *true* and *false*, any consistent and definite value gives us a satisfying result. This is also true for inconsistent and definite values, as they give us "all the information possible", and also present inconsistencies.

When modelling this setting, the refinement algorithm described in Chapter 4 can be used for distributive bilattices as well. We will not try to refine a definite value. Thus, only transitions or atomic propositions under some state with indefinite value will be suggested as failure node and reason.

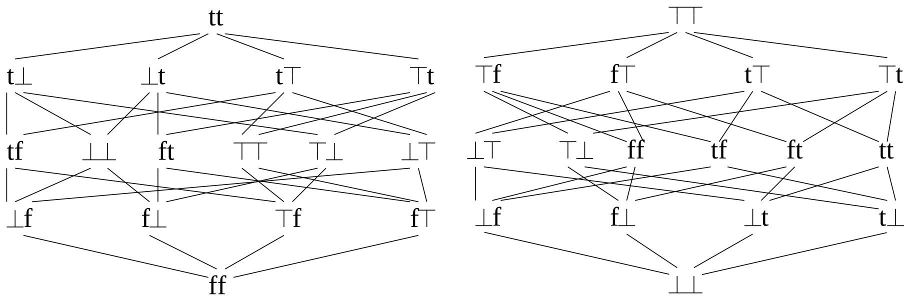
Example 7.1 *Figure 7.1 presents the bilattice of the* 4×4 *structure. This bilattice is created from the Belnap structure (Figure 3.1(a),(b)). The* 4×4 *structure represents two different views, each defined over the Belnap structure. Note that this structure is isomorphic to the* $\langle 2^{\{a,b\}} \times 2^{\{a,b\}} \rangle$ *structure (Figure 3.1(g),(h)). Consider the model M in Figure 7.2. For* $\varphi = \Diamond q$, $\|\varphi\|^M = (\bot t \wedge tf) \vee (\bot f \wedge tf) \vee (\top\top \wedge tt) = t\top$*. This is a result indicating that according to the first view the property holds, and according to the second view the property is inconsistent. The result is definite (and inconsistent), thus should not be refined. For* $\varphi = \Box p$, $\|\varphi\|^M = (\bot f \vee tt) \wedge (\bot t \vee ff) \wedge (\top\top \vee \bot\bot) = \bot t$*. This result is not definite. Based on the refinement algorithm presented in Chapter 4, the failure node is the node* $s_0 \vdash \Box p$*, and the failure reason is the edge to the node marked* $s_2 \vdash p$*. This edge corresponds to the transition* $R(s_0, s_2)$ *whose value is* $\bot f$*. Indeed, this is an indefinite transition, which needs to be refined.*

7.2 Models over Full Bilattices - STE and YASM

Symbolic Trajectory Evaluation (STE) [50]

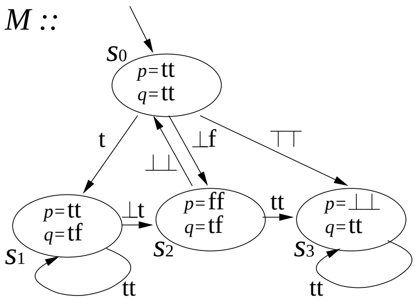
Symbolic Trajectory Evaluation (STE) is a symbolic simulation of hardware circuits with abstraction.

A hardware *circuit* C is a directed graph. The graph's nodes $\mathcal{N}$ are primary inputs and internal nodes, where internal nodes are latches and combinational gates. A node can get a boolean value, and a combinational



(a) (b)

Figure 7.1: Truth (a) and information (b) orders of 4 × 4 structure



(a)

Figure 7.2: Model M over 4 × 4 structure

gate represents a boolean operator. Given a directed edge (n_1, n_2) in a circuit, we say that n_1 is an *input* of n_2. The value of a node n is the result of applying its operator on its inputs at each clock cycle. The graph of C may contain loops, but not combinational loops. A circuit is formally defined as $C = \langle V, I_0, PI, F\rangle$, where V is the set of latches in C, I_0 is a set of possible initial values to V, PI is the set of primary inputs, and F is a set of transition functions such that for every $v_i \in V$, $f_i : 2^V \times 2^{PI} \rightarrow \{0, 1\}$.

In STE, a node can get a value in the Belnap structure, $\mathcal{B}(\mathcal{D})$= $(\{\top, \bot, t, f\}, \leq_t, \leq_i, \neg)$ (Figure 3.1(a),(b)). A node whose value cannot be determined by its inputs is given the value $\bot$. $\top$ is used to describe an over constrained node. This might occur when there is a contradiction between an external assumption on the circuit and its actual behavior. Under STE, F, the set of transition functions is defined such that for every $v_i \in V$, $f_i : \{\top, \bot, t, f\}^V \times \{\bot, t, f\}^{PI} \rightarrow \{\top, \bot, t, f\}$. Note that inputs cannot get a value $\top$, contradiction can occur only on the value of the latches.

A circuit can be viewed as a multi-valued Kripke structure $M = \langle \mathcal{L}, AP, S, s_0, R, \Theta\rangle$ where $\mathcal{L}$ is the Belnap structure and $AP = V \cup PI$. A *state* s in M is an assignment of values to every latch and every input, for every $v \in V$, $s(v) \in \{\top, \bot, t, f\}$ and for every $in \in PI$, $s(in) \in \{\bot, t, f\}$. The transition relation R is given by F: $R(s, s') \Leftrightarrow \forall v_i \in V[s'(v_i) = f_i(s\,|_V, s'\,|_{PI})]$. Where $s\,|_V$ refers to the value of the latches in s, and $s'\,|_{PI}$ refers to the value of the inputs in s'. If such assignment does not exist, then $R(s, s') = f$. Θ is defined such that for every $a \in AP$, $\Theta(a)(s) = s(a)$.

Due to the way STE (abstract) models are created, the Kripke models which are constructed in order to verify STE assertions have the following characteristics:

- Although models are defined over the Belnap structure, only atomic propositions can be evaluated to any of the four values $\top, \bot, t, f$.
- Transitions are evaluated to either t or f. Intuitively, all transitions are “must and may” transitions. There is no meaning in STE to transitions with the value $\bot$.
- Inputs are evaluated over 3 values: an input $in \in PI$ can be t,f or $\bot$, and cannot be evaluated to $\top$.
- Both the concrete and abstract models are defined over the Belnap structure. This results from the fact that inconsistencies (evaluation of some v_i to $\top$), which occur if there is an over constrained node, can

occur also in the concrete model.

YASM - a Software Model-Checker [36]

The authors in [36] present a framework for model checking programs via Kripke models over the Belnap structure. In their model, atomic propositions in states are evaluated to t, f or $\bot$ (a regular 3-valued semantics), whereas transitions are evaluated to t, f ,$\bot$ or $\top$. Intuitively, "may" transitions are evaluated to $\bot$, "may and must" transitions are evaluated to t and "must but not may" transitions are evaluated to $\top$. Their approach is based on treating unknowns resulting from abstraction differently from unknowns resulting from the environment. This approach is implemented via a symbolic software model checker called YASM.

Example 7.2 *[36] Figure 7.3 presents an example for a (partial) model created for the program $y := y-1$. The atomic propositions are the predicates $y > 2$ and $x = 2$. Note that the transition $R(s_0, s_0)$ has the value t, whereas transitions from state s_1 have the value $\bot$ to both s_0 and s_1. This is a result of the abstraction. If $y \leq 2$, then after the execution of $y := y - 1$ it is guaranteed that $y \leq 2$, on the other hand, if $y > 2$ then after executing $y := y - 1$ the value of y is not guaranteed. The fact that the value of x does not change is represented in the $\top$ transitions to s_3. The transitions from both s_0 and s_1 (where $x = 2$) to s_3 indicate that after execution of the program, the program "must" be in a state where $x = 2$. Based on the construction of the model described in [36], transitions to states with atomic propositions evaluated to $\bot$ will be given either a value of f or a value of $\top$.*

The authors prove that given a program and its corresponding abstraction, for every L_μ formula φ, the evaluation of φ on the abstract model will never be $\top$. Moreover, if the evaluation is t or f on the abstract model then its evaluation on the program is also t or f, respectively. We can then conclude that though mixed simulation is not defined between the concrete program and its abstraction, preservation of L_μ formulas holds.

Note that although the multi-valued Kripke model created by the abstraction is defined over the Belnap structure, the atomic propositions and also formulas are never evaluated to $\top$.

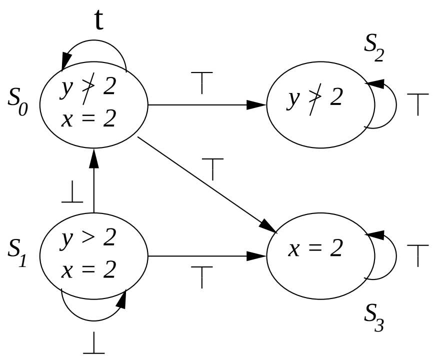


Figure 7.3: A 4-valued model

7.3 Compositional Model Checking of Multi-Valued Models Over Bilattices

We now discuss the compositional model checking framework when the models are defined over a bilattice rather then a CPDB. Recall that when defining the composition of two multi-valued models, we first defined which states might be composed. The goal is to compose states whose composition represents a state in the composed concrete system. In order to do so, we separated the states to weakly composable and composable states.

As discussed in Section 7.1, the connection between the concrete and abstract models for full bilattices is not defined through mixed simulation, but rather based on the specific abstraction. Thus, there is no general way to define composition of such models. As a result, the composition of models should be done with respect to the connection between the concrete and abstract model. We will define composition separately for the two cases we present here (STE and YASM).

STE

As discussed above, STE models are defined over the Belnap structure, but only latches can be evaluated to $\top$. Transitions are evaluated to t or f. In STE, the models synchronize on the value of the inputs (and not the latches). The reason is that the value of the latches is determined by the value of the latches in the *previous* stage, and by the value of the inputs in the *current* state, and also since the set of transition functions used (F) is the same for both models. An exception to that are the initial states which must synchronize on the value of the latches as well.

Recall that transitions in both the abstract and the concrete systems are always "may and must" transitions. Another characteristic of the system is that an abstract state in which some input has the value $\perp$ represents two concrete states (which must exist in the concrete model), one for which the input is t and the other for which the input is f. As a result, states which are weakly composable (in the abstract system) always represent a state in the concrete system. Thus, the weakly composable states can be treated as composable.

Following the above, the definitions of weakly composable states (Definition 6.1) and of composable states (6.2) should be replaced with the following:

Definition 7.3 *Let $s_1 \in S_1$, $s_2 \in S_2$ be states in M_1 and M_2 respectively. Then s_1, s_2 are* composable *if for every $p \in (AP_1 \cap PI_1) \cap (AP_2 \cap PI_2)$: $\Theta_1(p)(s_1) \oplus \Theta_2(p)(s_2)$ is consistent.*

We say that two models M_1 and M_2 are composable if their initial states agree on the value of their joint labeling. I.e., if for every $p \in AP_1 \cap AP_2$, $\Theta_1(p)(s_0^1) \oplus \Theta_2(p)(s_0^2)$ is consistent.

Composition of STE models is defined as in Definition 6.4, with the exception that states cannot be weakly composable and not composable.

The definition of the lifted model, $M_i\!\uparrow$ (Definition 6.7) should be modified for the specific abstraction. The definition of the transition mapping should be $R_i \uparrow = R_i$. This results from the fact that transitions in the abstract model all represent transitions in the concrete composed model. Under this definition for the lifted model, the mixed simulation relation between the composed model and the lifted model exists. Thus, $M_1 || M_2 \preceq M_i\!\uparrow$ (similar to Theorem 6.9).

The rest of the framework, as presented in Chapter 6, remains the same.

YASM

As described above, the atomic propositions are evaluated over t, f and $\perp$, thus the computation of weakly composable and composable states should be done over the 3-valued semantics. On the other hand, the evaluation of the transition relation is done over the Belnap structure, and the definition of composition is similar to the composition of models over CPDBs (Definition 6.4). In the composed model, transitions evaluated to $\top$ will be transitions to composable states, whose transitions in the composed system were $\top$.

In order to describe the compositional framework, we first have to define lifting of models (Definition 6.7) such that mixed simulation exists between the composed and lifted models. Thus, the following should hold: $M_1||M_2 \preceq M_i\uparrow$ (similar to Theorem 6.9). The definition of the states in the lifted model is similar to Definition 6.4. The definition for the transition relation is replaced by the following:
For each $s_i, t_i \in S_i$:

- If $R_i(s_i, t_i) \neq \top$ then $R_i\uparrow(s_i, t_i) = R_i(s_i, t_i) \wedge \bot$.
- Else (if $R_i(s_i, t_i) = \top$) then $R_i\uparrow(s_i, t_i) = \bot$.

Under the above definition for lifted models, it is indeed true that $M_1||M_2 \preceq M_i\uparrow$. Note that there are no transitions evaluated to $\top$ in the lifted model. Indeed, when model checking one component, we cannot guarantee that there "must" be some transition to a state, since the transition can be removed in the composed system.

The rest of the compositional framework remains the same. Note that in the definition of the product graph (Definition 6.14), the value of the edges is identical to their value in the composed model. Thus, we might have edges with the value $\top$ in the product graph.

Chapter 8

Conclusion

In this work we describe a *framework* for multi-valued model checking of L_μ formulas with respect to systems composed of several components, based on multi-valued abstraction and refinement.

We have considered bilattices as part of our framework. Based on the information order of a bilattice, we defined a mixed simulation relation over multi-valued models, preserving μ-calculus specifications. Bilattices and the mixed simulation relation allowed us to naturally define abstraction of models in the multi-valued context, and to describe the connection between concrete and abstract multi-valued models.

We have presented an automatic abstraction-refinement algorithm for multi-valued systems. To the best of our knowledge, this is the first abstraction-refinement algorithm defined to handle the general case of multi-valued systems.

Based on multi-valued abstraction and refinement, we presented our compositional framework, which can be described as follows.

- Lift each individual component M_i into a component $M_i\uparrow$ such that $M_1||M_2 \preceq M_i\uparrow$.
- Model check each of the lifted models separately. If the result is definite, then this also holds for the full system.
- Construct the product graph of the individual mc-graphs and model check it correctly.
- If the result on the product graph is definite, then this result holds for the full system. Otherwise, refine the components as needed.

We showed how our framework can be implemented for model checking of CPDBs, and alternation-free L_μ formulas. We applied a specific model checking and reason finding algorithm (Algorithm 2.12 and Sec. 4), but these can be replaced by other algorithms.

Our framework is suitable for full L_μ, provided that the model checking and reason finding algorithm can handle the full L_μ. Examples of such algorithms for a 3-valued structure can be found in [32]. Indeed, in [54] a compositional framework such as ours has been presented for the full $L_\mu$for a 3-valued logic.

We have discussed how our framework can be used for multi-valued structures that are described as full distributive bilattices. We presented our framework for two specific applications where models are defined over the Belnap structure, STE and YASM.

Our framework can also be used for logics other than the μ-calculus. For example, the *full-PML* logic, which extends the modal operators with past operators, AY and EY, is used in [3], along with a 6-valued structure (described in Figure 3.1(c),(d)). The structure is a CPDB, but since they use a logic with significantly different semantics, specific adaptations in some of the framework stages are needed.

Bibliography

[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[2] Alexander Asteroth, Christel Baier, and Ulrich Aßmann. Model checking with formula-dependent abstract models. In *Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 155–168, Paris, France, July 2001.

[3] Thomas Ball, Orna Kupferman, and Greta Yorsh. Abstraction for falsification. In *Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*, pages 67–81, Edinburgh, Scotland, UK, July 2005.

[4] Nuel .D Belnap. A useful four-valued logic. In G. Epstein and J. Dunn, editors, *Modern uses of multiple valued logics*, pages 8–37. D. Reidel, Dordrecht, 1977.

[5] Mihaela Gheorghiu Bobaru, Corina S. Pasareanu, and Dimitra Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In *Computer Aided Verification (CAV'08)*, volume 5123 of *LNCS*, Princeton, NJ, USA, July 2008. Springer.

[6] Glenn Bruns and Patrice Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 274–287, Trento, Italy, July 1999.

[7] Glenn Bruns and Patrice Godefroid. Temporal logic query checking. In *Logic in Computer Science (LICS'01)*, pages 409–417, Boston, Massachusetts, USA, June 2001. IEEE.

[8] Glenn Bruns and Patrice Godefroid. Model checking with multi-valued logics. In *International Colloquium for Automata, Languages and Programming (ICALP'04)*, volume 3142 of *LNCS*, pages 281–293, Turku, Finland, July 2004.

[9] Sagar Chaki and Ofer Strichman. Optimized L*-based assume-guarantee reasoning. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *LNCS*, Braga, Portugal, March 2007.

[10] William Chan. Temporal-logic queries. In *Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 450–463, Chicago, IL, USA, July 2000.

[11] Marsha Chechik, Benet Devereux, Steve Easterbrook, and Arie Gurfinkel. Multi-valued symbolic model-checking. *ACM Transactions on Software Engineering and Methodology*, 12:2003, 2003.

[12] Marsha Chechik, Benet Devereux, Steve Easterbrook, Albert Y. C. Lai, and Victor Petrovykh. Efficient multiple-valued model-checking using lattice representations. In *Concurrency Theory (CONCUR'01)*, volume 2154 of *LNCS*, pages 451–465, Aalborg, Denmark, August 2001.

[13] Marsha Chechik, Benet Devereux, Arie Gurfinkel, and Steve Easterbrook. Multi-valued symbolic model-checking. Technical Report CSRG-448, University of Toronto, April 2002.

[14] Marsha Chechik, Steve Easterbrook, and Victor Petrovykh. Model-checking over multi-valued logics. In *Formal Methods Europe (FME'01)*, volume 2021 of *LNCS*, pages 72–98, Berlin, Germany, March 2001.

[15] Marsha Chechik, Arie Gurfinkel, and Benet Devereux. chi-chek: A multi-valued model-checker. In *Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 505–509, Copenhagen, Denmark, July 2002.

[16] Yu-Fang Chen, Azadeh Farzan, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Learning minimal separating DFA's

for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'09)*, volume 5505 of *LNCS*, York, UK, March 2009.

[17] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. *Journal of the ACM*, 50(5):752–794, 2003.

[18] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT press, 1999.

[19] Jamieson M. Cobleigh, George S. Avrunin, and Lori A. Clarke. Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. *ACM Transactions on Software Engineering and Methodology*, 17(2), 2008.

[20] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Pasareanu. Learning assumptions for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, Warsaw, Poland, April 2003.

[21] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Principles of Programming Languages (POPL'77)*, pages 238–252, Los Angeles, CA, USA, January 1977.

[22] Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(2):253–291, March 1997.

[23] Luca de Alfaro, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Roberto Segala. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 395–410, Berlin, Germany, March 2000.

[24] Steve Easterbrook and Marsha Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *International Conference on Software Engineering (ICSE'01)*, pages 411–420, Toronto, Ontario, Canada, May 2001.

[25] Azadeh Farzan, Yu-Fang Chen, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. Extending automated compositional verification to the full class of omega-regular languages. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*, volume 4963 of *LNCS*, Budapest, Hungary, March 2008.

[26] Melvin Fitting. Bilattices are nice things, 2002.

[27] Mihaela Gheorghiu, Dimitra Giannakopoulou, and Corina S. Pasareanu. Refining interface alphabets for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *LNCS*, Braga, Portugal, March 2007.

[28] Dimitra Giannakopoulou and Corina S. Păsăreanu. Special issue on learning techniques for compositional reasoning. *Formal Methods in System Design*, 32(3):173–174, 2008.

[29] Patrice Godefroid and Radha Jagadeesan. Automatic abstraction using generalized model checking. In *Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 137–150, Copenhagen, Denmark, July 2002.

[30] Patrice Godefroid and Radha Jagadeesan. On the expressiveness of 3-valued models. In *Verification, Model Checking, and Abstract Interpretation (VMCAI'03)*, volume 2575 of *LNCS*, pages 206–222, New York, NY, USA, January 2003.

[31] Orna Grumberg, Martin Lange, Martin Leucker, and Sharon Shoham. Don't know in the μ-calculus. In *Verification, Model Checking, and Abstract Interpretation (VMCAI'05)*, volume 3385 of *LNCS*, pages 233–249, Paris, France, January 2005.

[32] Orna Grumberg, Martin Lange, Martin Leucker, and Sharon Shoham. When not losing is better than winning: Abstraction and refinement for the full μ-calculus. *Information and Computation*, 205(8):1130–1148, 2007.

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

[34] Anubhav Gupta, Kenneth L. McMillan, and Zhaohui Fu. Automated assumption generation for compositional verification. *Formal Methods in System Design*, 32(3):285–301, 2008.

[35] Arie Gurfinkel and Marsha Chechik. Multi-valued model checking via classical model checking. In *Concurrency Theory (CONCUR'03)*, volume 2761 of *LNCS*, pages 263–277, Marseille, France, September 2003.

[36] Arie Gurfinkel and Marsha Chechik. Why waste a perfectly good abstraction? In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 212–226, Vienna, Austria, April 2006.

[37] Arie Gurfinkel, Benet Devereux, and Marsha Chechik. Model exploration with temporal logic query checking. In *Foundations of Software Engineering (FSE'02)*, pages 139–148, Charleston, SC, USA, November 2002.

[38] Arie Gurfinkel, Ou Wei, and Marsha Chechik. Systematic construction of abstractions for model-checking. In *Verification, Model Checking, and Abstract Interpretation (VMCAI'06)*, volume 3855 of *LNCS*, pages 381–397, Charleston, SC, USA, January 2006.

[39] Michael Huth, Radha Jagadeesan, and D. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *European Symposium on Programming (ESOP'01)*, volume 2028 of *LNCS*, pages 155–169, Genova, Italy, April 2001.

[40] Michael Huth and Shekhar Pradhan. Lifting assertion and consistency checkers from single to multiple viewpoints. Technical Report 2002/11, Dept. of Computing, Imperial College, London, 2002.

[41] Cliff B. Jones. Specification and design of (parallel) programs. In *IFIP Congress*, pages 321–332, 1983.

[42] Beata Konikowska and Wojciech Penczek. Reducing model checking from multi-valued CTL* to CTL*. In *Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 226–239, Brno, Czech Republic, August 2002.

[43] Orna Kupferman and Yoad Lustig. Latticed simulation relations and games. In *Automated Technology for Verification and Analysis (ATVA'07)*, volume 4762 of *LNCS*, pages 316–330, Tokyo, Japan, October 2007.

[44] Robert P. Kurshan. *Computer-Aided Verification of coordinating processes - the automata theoretic approach.* Princeton University Press, Princeton, New Jersey, 1994.

[45] Jørn Lind-Nielsen and Henrik Reif Andersen. Stepwise CTL model checking of state/event systems. In *Computer Aided Verification (CAV'99)*, volume 1633 of *LNCS*, pages 316–327, Trento, Italy, July 1999.

[46] Wonhong Nam, P. Madhusudan, and Rajeev Alur. Automatic symbolic compositional verification by learning assumptions. *Formal Methods in System Design*, 32(3):207–234, 2008.

[47] Abelardo Pardo and Gary D. Hachtel. Automatic abstraction techniques for propositional μ-calculus model checking. In *Computer Aided Verification (CAV'97)*, volume 1254 of *LNCS*, pages 12–23, Haifa, Israel, June 1997.

[48] Corina S. Pasareanu, Dimitra Giannakopoulou, Mihaela Gheorghiu Bobaru, Jamieson M. Cobleigh, and Howard Barringer. Learning to divide and conquer: applying the L* algorithm to

automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3), 2008.

[49] Amir Pnueli. In transition from global to modular temporal reasoning about programs. pages 123–144, 1985.

[50] Carl-Johan H. Seger and Randal E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. *Formal Methods in System Design*, 6(2), 1995.

[51] Sharon Shoham. A game-based framework for CTL counterexamples and abstraction-refinement. Master's thesis, Dept. of Computer Science, Technion - Israel Institute of Technology, 2003.

[52] Sharon Shoham. *Abstraction-Refinement and Modularity in μ-Calculus Model Checking.* PhD thesis, Department of Computer Science, Technion - Israel Institute of Technology, 2009.

[53] Sharon Shoham and Orna Grumberg. Multi-valued model checking games. In *Automated Technology for Verification and Analysis (ATVA'05)*, volume 3707 of *LNCS*, pages 354–369, Taipei, Taiwan, October 2005.

[54] Sharon Shoham and Orna Grumberg. Compositional verification and 3-valued abstractions join forces. In *Static Analysis Symposium (SAS'07)*, volume 4634 of *LNCS*, pages 69–86, Kongens Lyngby, Denmark, August 2007.

[55] Nishant Sinha and Edmund M. Clarke. SAT-based compositional verification using lazy learning. In *Computer Aided Verification (CAV'07)*, volume 4590 of *LNCS*, pages 39–54, Berlin, Germany, July 2007.

[56] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math*, 5:285–309, 1955.