A Framework For Compositional Verification of Multi-Valued Systems Via Abstraction-Refinement[☆]

Yael Meller[a], Orna Grumberg[a], Sharon Shoham[b]

[a] *Technion – Israel Institute of Technology*
[b] *Academic College of Tel Aviv Yaffo*

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.

In a multi-valued model of a system, both the transitions between system states and the labeling of states are assigned values from a given lattice. Such 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.

We formalize our framework based on bilattices, consisting of a truth lattice and an information lattice. Formulas are interpreted, as usual, on the truth lattice. The information lattice determines how *definite* the value is, in terms of the concrete system being modeled.

Our compositional approach model checks individual components of a system, where each component is viewed as an *abstraction* of the entire system. Only if all individual checks return *indefinite* values (i.e., values that cannot be carried over to the composed system), 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, this means that the composed system is in itself too abstract, and a *refinement* of the multi-valued system is needed. Refinement is now aimed at increasing the information level of model details, thus also increasing the information level of the model checking result.

We suggest how multi-valued models, and their underlying bilattices, should be composed, checked, and refined. Our approach is based on multi-valued model checking graphs, which can be thought of as a product of the model and the formula to be checked.

Keywords: multi-valued model checking, mu-calculus, bilattices, mixed simulation, refinement, compositional model checking

[☆] A preliminary version of this paper appeared in [49].

Email addresses: ymeller@cs.technion.ac.il (Yael Meller), orna@cs.technion.ac.il (Orna Grumberg), sharon.shoham@gmail.com (Sharon Shoham)

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 concrete systems by distinguishing between several levels of uncertainty and inconsistency [5, 10, 6, 24, 39, 37]. These models have been widely used for abstraction as well [54, 55, 2, 31, 36, 32].

For example, 3-valued models are used to describe models with partial information [5], using *true* and *false* to model known values, and *indefinite* to model an unknown value. Such models can also be the result of *abstraction* which collapses together multiple states. In case one state has an outgoing transition in the concrete system and another does not, the transition will have an indefinite value in the abstract (3-valued) model. 4-valued models can model disagreement and their generalizations are used to handle inconsistent views of a system [24, 39], by considering tuples of values, where each index in the tuple represents one view. Temporal logic query checking [10, 6, 37] can also be reduced to multi-valued model checking.

Multi-valued models, both abstract and concrete, may still suffer from the *state explosion problem*. Two of the most successful approaches for fighting this problem in classical (2-valued) model checking are abstraction-refinement and compositional verification. *Abstraction-refinement* is an iterative process, in which a model is abstracted by removing or simplifying details, model checked, and if an inconclusive result is obtained due to the abstraction, the abstract model is refined by adding more details into it. In *compositional* model checking, parts of the system are verified separately in order to avoid the construction of the entire system. Typically, some information needs to be exchanged between these checks in order to enable verification of the system.

In this work, we develop a compositional multi-valued model checking based on abstraction and refinement.

The first step we take in formalizing our 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 labeling states and values of transitions between states 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 (which combines existential and universal quantifiers), a bidirectional correspondence between transitions of M_1 and M_2 is needed. To capture this bidirectional correspondence, 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 represent conclusive results and 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 Section 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 [58]. 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 composed system is in itself abstract, the check of the partially composed system might still be indefinite, in which case 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 [57]. 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 from a given concrete one.

Other works deal with several aspects of multi-valued model checking (as discussed in Section 8), but to the best of our knowledge none investigates a compositional approach. Our framework for compositional multi-valued model checking is applicable to any multi-valued model defined over a CPDB. We also show the applicability of our approach to multi-valued models defined over full distributive bilattices, with STE [54] and YASM [36] as concrete examples. We

consider specifications given as μ-calculus formulas, but with certain adaptations, different logics can be handled as well.

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. Organization

The rest of the paper is organized as follows. In the next section we give the necessary background for multi-valued models, including the multi-valued μ-calculus, and a multi-valued model checking algorithm [57]. In Section 3 we describe bilattices, define Consistent Partial Distributive Bilattices and present their attributes. We present mixed simulation for multi-valued models in Section 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 Section 4 as well. We then continue with investigation of properties of partial model checking graphs, used by our compositional approach, in Section 5. We describe composition of multi-valued models, and present our compositional framework in Section 6. As the models at hand might be abstract, our compositional framework includes use of the abstraction-refinement developed in Section 4. In Section 7 we discuss our framework with respect to full bilattices, and in Section 8 we survey related work. Finally, we discuss some conclusions in Section 9.

2. Preliminaries

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

Definition 2.1. *A* complete lattice $\mathcal{L}=(L,\leq)$ *consists of a set* L *with a partial order* $\leq$ *over* L*, where every 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* $(a \sqcup b) \sqcap c = (a \sqcap c) \sqcup (b \sqcap c)$*.*

All complete lattices have a greatest (top) element and a least (bottom) element. From here on we refer to complete lattices as *lattices*.

Examples of lattices are shown in Figure 1(a),(b),(c),(e),(g) and (h) on page 6. Note that the partially ordered sets presented in Figure 1(d) and (f) are not lattices, since in both cases a least upper bound does not always exist (e.g., for T and M in (d), and for FF and TF in (f)).

Definition 2.2. *A* De Morgan algebra *is a structure* $\mathcal{D}=(L,\leq,\neg)$*, where* $(L,\leq)$ *is a finite distributive lattice,* $\neg : L \to L$ *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* $\neg(a \sqcup b) = \neg a \sqcap \neg b$*.*

The greatest (top) element of a De Morgan algebra is denoted $true$, and its least (bottom) element is denoted $false$.

2.1. Multi-Valued Models and μ-calculus

Definition 2.3. *A* Multi-Valued Kripke model *is a 6-tuple* $M = \langle \mathcal{D}, AP, S, s_0, R, \Theta \rangle$*, where:*

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

The R component of a multi-valued Kripke model generalizes the traditional notion of a transition relation, which defines the set of transitions of a system as pairs of states. A (2-valued) transition relation can be viewed as a mapping from pairs of states to Boolean values indicating if the pair represents a transition or not. In the multi-valued case, a value from L is used instead of a Boolean value. Θ generalizes the traditional labeling function in a similar way.

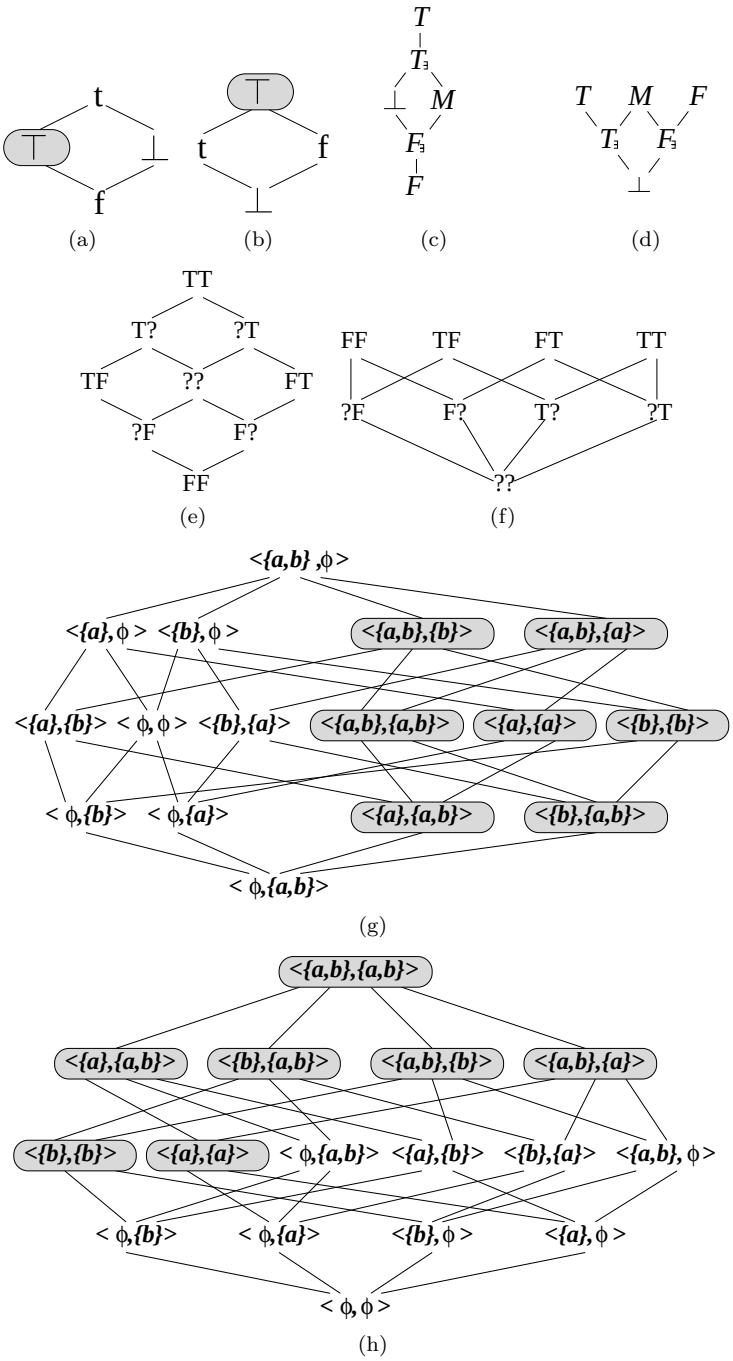


Figure 1: Truth (a) and information (b) orders of 4-valued Belnap structure; Truth (c) and information (d) orders of 6-valued structure; Truth (e) and information (f) orders of 3×3 structure; Truth (g) and information (h) orders of $\langle 2^{\{a,b\}}, 2^{\{a,b\}} \rangle$ structure; Boxed nodes are inconsistent

Definition 2.4. *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.

Intuitively, in the traditional (2-valued) case, the modal operator $\Diamond$ stands for "all successors", whereas $\Box$ stands for "exists a successor". μ denotes a least fixpoint, whereas ν denotes greatest fixpoint. Least fixpoints are used to express liveness, while greatest fixpoints express safety properties. For example, the L_μ formula $\mu Z.(p \vee \Diamond Z)$ expresses the (existential) liveness property "there exists a path along which p eventually holds". Dually, $\nu Z.(p \wedge \Box Z)$ expresses the (universal) safety property "p is true in all the reachable states". Formally, the 2-valued semantics of a formula determines which states satisfy the formula. Equivalently, it can be viewed as a mapping of states to Boolean values determining for each state whether it satisfies the formula. The multi-valued semantics of L_μ is defined next.

The multi-valued semantics of a μ-calculus formula φ in a multi-valued Kripke model M over $\mathcal{D}{=}\,(L, \leq, \neg)$ is a mapping of the model's states to truth values from L describing the truth value of φ in each state. Since L_μ formulas might contain free variables (which are not bound by any fixpoint quantifier), we need to introduce a notion of a (multi-valued) environment which provides the interpretation of free variables.

An *environment* $\mathcal{V} : Var \to (S \to L)$, where S is the set of states of M, maps each variable to a mapping of states to truth values in L. 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 now turn to the formal definition of the multi-valued semantics of μ-calculus [7]. Given a multi-valued Kripke model $M = \langle \mathcal{D}, AP, S, s_0, R, \Theta \rangle$ and environment $\mathcal{V}$, the multi-valued interpretation of a formula φ w.r.t. M and $\mathcal{V}$, denoted $\|\varphi\|^M_\mathcal{V}$, is a mapping $S \to L$, defined below. In the definition, lfp, gfp stand for least and greatest fixpoints respectively, which exist based on [60]. $\|\varphi\|^M_\mathcal{V}$ is defined inductively as follows:

$$\|p\|^M_\mathcal{V} = \lambda s.\Theta(p)(s)$$
$$\|\neg p\|^M_\mathcal{V} = \lambda s.\neg\Theta(p)(s)$$
$$\|\varphi_1 \vee \varphi_2\|^M_\mathcal{V} = \lambda s.\|\varphi_1\|^M_\mathcal{V} \sqcup \|\varphi_2\|^M_\mathcal{V}$$

$$\|\varphi_1 \wedge \varphi_2\|^M_{\mathcal{V}} = \lambda s. \|\varphi_1\|^M_{\mathcal{V}} \sqcap \|\varphi_2\|^M_{\mathcal{V}}$$

$$\|\Diamond\varphi\|^M_{\mathcal{V}} = \lambda s. \bigsqcup_{s' \in S} (R(s,s') \sqcap \|\varphi\|^M_{\mathcal{V}}(s'))$$

$$\|\Box\varphi\|^M_{\mathcal{V}} = \lambda s. \bigsqcap_{s' \in S} (\neg R(s,s') \sqcup \|\varphi\|^M_{\mathcal{V}}(s'))$$

$$\|Z\|^M_{\mathcal{V}} = \mathcal{V}(Z)$$

$$\|\mu Z.\varphi\|^M_{\mathcal{V}} = \mathrm{lfp}(\lambda g. \|\varphi\|^M_{\mathcal{V}[Z=g]})$$

$$\|\nu Z.\varphi\|^M_{\mathcal{V}} = \mathrm{gfp}(\lambda g. \|\varphi\|^M_{\mathcal{V}[Z=g]})$$

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

In some cases, it is convenient to use a fixpoint characterization of μ-calculus formulas, given by approximants. Approximants of L_μ formulas are defined in the usual way:

Definition 2.5. *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 2.6. *[60] 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 $s \in S$, $\|\eta Z.\varphi\|^M_{\mathcal{V}}(s) = Z^\alpha(s)$.*

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 [57]. 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 [57], and the model checking algorithm that is based on it.

2.2.1. Model-Checking Graphs (mc-graphs)

Let $M = \langle \mathcal{D}, AP, S, s_0, R, \Theta \rangle$ be a multi-valued Kripke model over $\mathcal{D}= (L, \leq, \neg)$ 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 $(n, n') \in E$.

Terminal nodes in $G(M, \varphi_0)$ are assigned values from L, as follows:

Definition 2.7. *([57]) Let n be a terminal node in $G(M, \varphi_0)$. Then $val(n)$ 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 $val(t \vdash \Box\varphi) = true$.*

2.2.2. Model-Checking Algorithm For mc-graphs

In [57] an algorithm for computing a value of nodes on an mc-graph is presented. The algorithm handles the *alternation-free* fragment of L_μ, where no nesting of fixpoints is allowed. Given an mc-graph $G(M, \varphi_0) = (n^0, N, E)$, also called G in short, and a (partial) function $val : N \to L$ which maps terminal nodes in G to values in L (Definition 2.7), the algorithm returns an mc-function $\chi : N \to L$ that maps each node to a value from L.

Algorithm 2.8 (mc-algorithm [57]). *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. Such an order can be obtained by a*

topological sort of G, followed by an arbitrary completion of the partial order into a total order.
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, $temp(n)$, computed as follows. 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 iteratively apply the previous rules for $\vee$,$\wedge$-nodes in arbitrary order 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 [57], the connection between χ and the model checking problem is proved, by showing that $\chi(n^0) = \|\varphi_0\|^M(s_0)$. That is, the value that the **mc-algorithm** associates with the initial node ($\chi(n^0)$) equals the interpretation of φ_0 on the initial state of M ($\|\varphi_0\|^M(s_0)$). In the context of this work we will also be interested in the internal nodes of G and not only in the initial node. We therefore generalize the correspondence between χ and the multi-valued semantics to *all* nodes in G. This is captured in the following theorem.

For $\psi \in Sub(\varphi_0)$, ψ^* denotes 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 2.9. *Let $G(M, \varphi_0)$ be an 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 the nodes reachable from n.

Recall that the evaluation of the mc-graph is done bottom-up. That is, MSCCs are handled by increasing values of i, reflecting the reachability relation between nodes. Recall also that by the construction of the mc-graph, if ψ is a closed L_μ formula then all nodes $n = s \vdash \psi$ are part of trivial MSCCs. Therefore, if ψ is a closed L_μ formula then for all nodes $n = s \vdash \psi$, the handling of the subgraph induced by n by the **mc-algorithm** is the same as applying the **mc-algorithm** on $G(M', \psi)$ where M' is identical to M except that its initial state is s. By the correctness proof provided in [57], we can conclude that χ is correct for all nodes $n = s \vdash \psi$ such that ψ is a closed L_μ formula.

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 in additional transitions in G'

from nodes of the form $s' \vdash fp(Z)$ to nodes of the form $s' \vdash Z$. By definition of the updating of χ, these transitions do not change the value of the rest of the nodes. In particular, $\chi'(s \vdash \psi^*) = \chi(s \vdash \psi)$.

□

Let $G(M, \varphi_0)$ be an mc-graph. We say that $\chi : N \to L$ is *semantically correct* if for every $s \vdash \psi \in N$, $\chi(s \vdash \psi) = \|\psi^*\|^M(s)$. From Theorem 2.9 we can conclude that applying the **mc-algorithm** on $G(M, \varphi_0)$ results in an mc-function χ which is semantically correct.

3. Bilattices and Partial Bilattices

In this section we introduce bilattices, consider several of their attributes, and define the notion of partial bilattices. The use of bilattices (or partial bilattices) to present a multi-valued structure will then help us in defining 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* $\mathcal{B}=(B, \leq_{\mathbf{i}}, \leq_{\mathbf{t}}, \neg)$ *such that:*

1. $\mathcal{B}_i=(B, \leq_{\mathbf{i}})$ *is a lattice and* $\mathcal{B}_t=(B, \leq_{\mathbf{t}}, \neg)$ *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* $\leq_{\mathbf{i}}$ *and* $\leq_{\mathbf{t}}$*.*
3. *all meets and joins distribute over each other.*
4. *negation* $(\neg)$ *is* $\leq_{\mathbf{i}}$ *monotone.*

Note that requirement (4) defines the monotonicity of $\neg$ with respect to $\leq_{\mathbf{i}}$, whereas $\neg$ is anti-monotone with respect to $\leq_{\mathbf{t}}$ (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 $\leq_{\mathbf{t}}$ is an order on the "degree of truth". We refer to it as the *truth order of* $\mathcal{B}$. The bottom in this order is denoted by $false$ and the top by $true$. Thus $false \leq_{\mathbf{t}} x \leq_{\mathbf{t}} true$ for any $x \in B$. The meet and join operations for $\leq_{\mathbf{t}}$ are denoted by $\wedge$ and $\vee$ respectively. The relation $\leq_{\mathbf{i}}$ is an order on the "degree of information". Thus, if $x \leq_{\mathbf{i}} y$, y gives us at least as much information as x (and possibly more). We therefore call $\leq_{\mathbf{i}}$ the *information order of* $\mathcal{B}$. The meet and join operations for $\leq_{\mathbf{i}}$ are denoted $\otimes$ and $\oplus$ respectively. The bottom in the $\leq_{\mathbf{i}}$ order is denoted by $\bot$ 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* $(D \times D, \leq_{\mathbf{i}}, \leq_{\mathbf{t}}, \neg)$ *such that:*

- $\langle a, b\rangle \leq_{\mathbf{i}} \langle c, d\rangle \triangleq a \leq c$ *and* $b \leq d$
- $\langle a, b\rangle \leq_{\mathbf{t}} \langle c, d\rangle \triangleq a \leq c$ *and* $d \leq b$
- $\neg\langle a, b\rangle \triangleq \langle b, a\rangle$

Theorem 3.3. *[26] Let $\mathcal{D}$=$(D, \leq, \neg)$ be a De Morgan algebra. 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" (i.e., what is known w.r.t. $\leq_{\mathbf{t}}$) is x and whose "degree of falsity" (i.e., what is known w.r.t. the inverse of $\leq_{\mathbf{t}}$) is y. If we view $\mathcal{D}$ as a concrete truth domain, $\mathcal{B}(\mathcal{D})$ can be viewed as its abstraction. Given an element $c \in D$, $\langle x, y\rangle \in D \times D$ approximates c if x is equally or less true than c, and y is equally or less false than c. Thus, $\langle c, \neg c\rangle$ is the best approximation of c, and $\langle x, y\rangle$ approximates c if $\langle x, y\rangle \leq_{\mathbf{i}} \langle c, \neg c\rangle$. We say that $\langle x, y\rangle \in D \times D$ is *consistent* if $\langle x, y\rangle \leq_{\mathbf{i}} \langle c, \neg c\rangle$ for some $c \in D$, i.e., $\langle x, y\rangle \in D \times D$ approximates some $c \in D$. 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 $\langle c, \neg c\rangle \leq_{\mathbf{i}} \langle x, y\rangle$ 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 1(a),(b) on page 6 present an example of the distributive bilattice for the 4-valued Belnap structure [3]. 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 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 $\mathcal{B}(\mathcal{D})$.

The following lemma provides an explicit characterization of $\wedge, \vee, \otimes, \oplus$ in the induced bilattice $\mathcal{B}(\mathcal{D})$ in terms of $\sqcup$ and $\sqcap$ of $\mathcal{D}$.

Lemma 3.5. *Let $\mathcal{B}(\mathcal{D})$=$(D \times D, \leq_{\mathbf{i}}, \leq_{\mathbf{t}}, \neg)$ be the distributive bilattice induced by $\mathcal{D}$=$(D, \leq, \neg)$. 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 the claim 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:

- $\langle e, f\rangle \leq_{\mathbf{t}} \langle a, b\rangle$
- $\langle e, f\rangle \leq_{\mathbf{t}} \langle c, d\rangle$
- For every $\langle e', f'\rangle$ if $\langle e', f'\rangle \leq_{\mathbf{t}} \langle a, b\rangle$ and $\langle e', f'\rangle \leq_{\mathbf{t}} \langle c, d\rangle$ then $\langle e', f'\rangle \leq_{\mathbf{t}} \langle e, f\rangle$

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 $\mathcal{B}(\mathcal{D})$, $\langle a \sqcap c, b \sqcup d\rangle \leq_{\mathbf{t}} \langle a, b\rangle$ and $\langle a \sqcap c, b \sqcup d\rangle \leq_{\mathbf{t}} \langle c, d\rangle$. We can then conclude that $\langle a \sqcap c, b \sqcup d\rangle \leq_{\mathbf{t}} \langle e, f\rangle$.

By the definition of truth order on $\mathcal{B}(\mathcal{D})$, since $\langle e, f\rangle \leq_{\mathbf{t}} \langle a, b\rangle$ and $\langle e, f\rangle \leq_{\mathbf{t}} \langle c, d\rangle$, 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 $\langle e, f\rangle \leq_{\mathbf{t}} \langle a \sqcap c, b \sqcup d\rangle$.

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

Definition 3.6. *$\mathcal{P}$= $(L, \leq)$ is a* partial lattice *if it satisfies the requirements of a lattice, except that join is possibly not always defined. A* partial distributive bilattice *is a structure $\mathcal{P}$=$(B, \leq_{\mathbf{i}}, \leq_{\mathbf{t}}, \neg)$ that satisfies the requirements of a distributive bilattice (Definition 3.1), except that $\mathcal{P}_i$=$(B, \leq_{\mathbf{i}})$ is a partial lattice, and requirements (2) and (3) hold for join ($\oplus$) of $\mathcal{P}_i$ only if it is defined.*

Definition 3.7. *Let $\mathcal{B}(\mathcal{D})$=$\langle D \times D, \leq_{\mathbf{t}}, \leq_{\mathbf{i}}, \neg\rangle$ be a bilattice, and let $P \subseteq D \times D$ be the set of all consistent elements in $\mathcal{B}(\mathcal{D})$. Then $\mathcal{P}(\mathcal{B}(\mathcal{D}))$=$\langle P, \leq_{\mathbf{t}}, \leq_{\mathbf{i}}, \neg\rangle$ is the* consistent structure *induced by $\mathcal{B}(\mathcal{D})$, where $\leq_{\mathbf{t}}$, $\leq_{\mathbf{i}}$ and $\neg$ in $\mathcal{P}(\mathcal{B}(\mathcal{D}))$ 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. $\bot$, *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 $\mathcal{B}(\mathcal{D})$=$\langle D \times D, \leq_{\mathbf{t}}, \leq_{\mathbf{i}}, \neg\rangle$ be a bilattice, and let $a, b \in D \times D$ be consistent elements, then $a \wedge b$, $a \vee b$, $\neg a$ and $a \otimes b$ are consistent as well.*

Proof: Let $a = \langle x, y\rangle$ and $b = \langle z, w\rangle$. We prove this for each of the operators:

- $v = \neg\langle x, y\rangle$: By Definition 3.2, $v = \langle y, x\rangle$. $\langle x, y\rangle$ is consistent, thus $y \leq \neg x$. Since D is a De Morgan algebra we can conclude that $\neg y \geq \neg\neg x$, thus $x \leq \neg y$. This means that $\langle y, x\rangle$ is consistent as well.

- $v = \langle x, y\rangle \wedge \langle z, w\rangle$: By Lemma 3.5, $v = \langle x, y\rangle \wedge \langle z, w\rangle = \langle x \sqcap z, y \sqcup w\rangle$. $\langle x, y\rangle$ and $\langle z, w\rangle$ are both consistent, meaning: $y \leq \neg x$ and $w \leq \neg z$. As $\sqcup$ is monotone with respect to $\leq$ we can conclude that $(y \sqcup w) \leq (\neg x \sqcup \neg z)$. By De Morgan, this means that $(y \sqcup w) \leq \neg(x \sqcap z)$, thus v is consistent.

- $v = \langle x, y\rangle \vee \langle z, w\rangle$: By Lemma 3.5, $v = \langle x, y\rangle \vee \langle z, w\rangle = \langle x \sqcup z, y \sqcap w\rangle$. $\langle x, y\rangle$ and $\langle z, w\rangle$ are both consistent, meaning: $y \leq \neg x$ and $w \leq \neg z$. As $\sqcap$ is monotone with respect to $\leq$ we can conclude that $(y \sqcap w) \leq (\neg x \sqcap \neg z)$. By De Morgan, this means that $(y \sqcap w) \leq \neg(x \sqcup z)$, thus v is consistent.

- $v = \langle x, y\rangle \otimes \langle z, w\rangle$: By Lemma 3.5, $v = \langle x, y\rangle \otimes \langle z, w\rangle = \langle x \sqcap z, y \sqcap w\rangle$. $\langle x, y\rangle$ and $\langle z, w\rangle$ are both consistent, meaning: $y \leq \neg x$ and $w \leq \neg z$. As $\sqcup$ is monotone with respect to $\leq$ we can conclude that $(y \sqcup w) \leq (\neg x \sqcup \neg z)$. By De Morgan, this means that $(y \sqcup w) \leq \neg(x \sqcap z)$. We also know that $(y \sqcap w) \leq (y \sqcup w)$, 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 $\mathcal{B}(\mathcal{D})=\langle D\times D, \leq_{\mathbf{t}}, \leq_{\mathbf{i}}, \neg\rangle$ be a bilattice, and let $\mathcal{P}(\mathcal{B}(\mathcal{D}))=\langle P, \leq_{\mathbf{t}}, \leq_{\mathbf{i}}, \neg\rangle$ be the consistent structure induced by it, then $\mathcal{P}(\mathcal{B}(\mathcal{D}))$ is a partial distributive bilattice.*

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

Lemma 3.10. *Let $\mathcal{B}(\mathcal{D})=\langle D\times D, \leq_{\mathbf{t}}, \leq_{\mathbf{i}}, \neg\rangle$ be a bilattice, and let $\mathcal{P}(\mathcal{B}(\mathcal{D}))=\langle P, \leq_{\mathbf{t}}, \leq_{\mathbf{i}}, \neg\rangle$ 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 $\mathcal{B}(\mathcal{D})$ and $\mathcal{P}(\mathcal{B}(\mathcal{D}))$ are the same. Also, if the value of $a \oplus b$ is defined on $\mathcal{P}(\mathcal{B}(\mathcal{D}))$, then it is equal to the value of $a \oplus b$ on $\mathcal{B}(\mathcal{D})$ as well.*

Proof: We denote $\mathcal{B}(\mathcal{D})$ by $\mathcal{B}$. 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 $\leq_{\mathbf{i}}$, $\leq_{\mathbf{t}}$ and $\neg$ for $\mathcal{P}(\mathcal{B})$ are the same as for $\mathcal{B}$ when it is reduced to the consistent elements, and since $P(B)$ consists of all consistent elements in $D \times D$, then the values of $\leq_{\mathbf{i}}$, $\leq_{\mathbf{t}}$ and $\neg$ are the same for $\mathcal{B}$ and $\mathcal{P}(\mathcal{B})$.

For $a \oplus b$, by Lemma 3.5, if $a = \langle a_1, a_2\rangle$ and $b = \langle b_1, b_2\rangle$, then in $\mathcal{B}$, $a \oplus b = \langle a_1 \sqcup b_1, a_2 \sqcup b_2\rangle$. If this is a consistent value, and since $\mathcal{P}(\mathcal{B})$ includes all consistent elements, then this value will be the result in $\mathcal{P}(\mathcal{B})$ as well. Thus, the values in $\mathcal{P}(\mathcal{B})$ and in $\mathcal{B}$ 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) holds on $\mathcal{P}(\mathcal{B})$. Recall that the requirements of a partial

distributive bilattice are defined based on the requirements of a distributive bilattice (Definition 3.1), with several differences. We therefore consider each of the requirements listed in Definition 3.1 with the required adaptations (we split the first requirement into two):

1(a). We show that $\mathcal{P}_i$=$(P, \leq_{\mathbf{i}})$ is a partial lattice. That is, $\mathcal{P}_i$ consists of a set P with partial order $\leq_{\mathbf{i}}$ 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.

1(b). We show that $\mathcal{P}_t$=$(P, \leq_{\mathbf{t}}, \neg)$ 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 $\mathcal{B}_t$=$(D \times D, \leq_{\mathbf{t}}, \neg)$ is a De Morgan algebra, so is $\mathcal{P}_t$.

2. 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 $\leq_{\mathbf{i}}$ and $\leq_{\mathbf{t}}$, 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 $\leq_{\mathbf{i}}$ and $\leq_{\mathbf{t}}$ on $\mathcal{P}(\mathcal{B})$ are the same as on $\mathcal{B}$, reduced to consistent elements. Thus, the monotonicity applies here as well.
3. 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 $\mathcal{P}(\mathcal{B})$.
4. We show that negation ($\neg$) is $\leq_{\mathbf{i}}$ monotone. By Lemma 3.8, for $a \in D \times D$ a consistent element, $\neg a$ is consistent as well, and since $\leq_{\mathbf{i}}$ under $\mathcal{P}(\mathcal{B})$ is equal to its definition under $\mathcal{B}$, reduced to consistent elements, then negation is $\leq_{\mathbf{i}}$ 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 $\langle x, \neg x\rangle \geq_{\mathbf{i}} \langle x, y\rangle$.

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

Theorem 3.11. *Let $\mathcal{B}$= $\langle B, \leq_{\mathbf{t}}, \leq_{\mathbf{i}} \neg\rangle$ be 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, i.e. $\mathcal{B} = \mathcal{P}(\mathcal{B}(\mathcal{D}))$, $a \in B$ is definite if and only if $a = \langle c, \neg c\rangle$, for some $c \in D$. Thus, for proving the theorem, it is enough to prove the following statement: Let $\mathcal{B}$=$\langle D \times D, \leq_{\mathbf{t}}, \leq_{\mathbf{i}}, \neg\rangle$ be a bilattice (distributive

or CPDB), 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 $c, d \in D \times D$, then $a \wedge b$, $a \vee b$ and $\neg a$ are equal to $\langle e, \neg e \rangle$ for some $e \in D \times D$.

We prove both parts of the statement above according to the different operators:

- $v = \neg\langle x, y \rangle$: By definition of negation $v = \langle y, x \rangle$ (Definition 3.2). Since we assume $\langle x, y \rangle$ is definite, then $y \geq \neg x$. Based on De Morgan, $x \geq \neg y$, thus v is definite as well.

  For $v = \neg\langle x, \neg x \rangle$ we get $v = \langle \neg x, x \rangle$, which, clearly, is of the required form.

- $v = \langle x, y \rangle \wedge \langle w, z \rangle$: By Lemma 3.5, $v = \langle x \sqcap w, y \sqcup z \rangle$. We assume $y \geq \neg x$ and $z \geq \neg w$, then since the $\sqcup$ operator is monotone with respect to $\leq$, $y \sqcup z \geq \neg x \sqcup \neg w$. By De Morgan $y \sqcup z \geq \neg(x \sqcap w)$, thus v is definite.

  For $v = \langle x, \neg x \rangle \wedge \langle w, \neg w \rangle$ we get $v = \langle x \sqcap w, \neg x \sqcup \neg w \rangle$, and by De Morgan, this means that $v = \langle x \sqcap w, \neg(x \sqcap w) \rangle$, which is of type $\langle e, \neg e \rangle$ for $e = x \sqcap w \in D$.

- $v = \langle x, y \rangle \vee \langle w, z \rangle$: By Lemma 3.5, $v = \langle x \sqcup w, y \sqcap z \rangle$. We assume $y \geq \neg x$ and $z \geq \neg w$, then since the $\sqcap$ operator is monotone with respect to $\leq$, $y \sqcap z \geq \neg x \sqcap \neg w$. By De Morgan $y \sqcap z \geq \neg(x \sqcup w)$, thus v is definite.

  For $v = \langle x, \neg x \rangle \vee \langle w, \neg w \rangle$ we get $v = \langle x \sqcup w, \neg x \sqcap \neg w \rangle$, and by De Morgan, this means that $v = \langle x \sqcup w, \neg(x \sqcup w) \rangle$, which is of type $\langle e, \neg e \rangle$ for $e = x \sqcup w \in D$.

□

Example 3.12. *Examples of CPDBs appear in Figure 1 on page 6. The CPDB induced by the bilattice of the* Belnap structure *is described in Figure 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 [42], where* $? \triangleq \bot$, $T \triangleq t$ *and* $F \triangleq f$.

The structure 3×3 is defined by the CPDB in Figure 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* $\langle a_1, a_2 \rangle$ *and* $\langle b_1, b_2 \rangle$ *in* $\{T, F\} \times \{T, F\}$, $\langle a_1, a_2 \rangle \leq \langle b_1, b_2 \rangle$ *iff* $a_1 \leq b_1$ *and* $a_2 \leq b_2$. *Also,* $\neg\langle a_1, a_2 \rangle \triangleq \langle \neg a_1, \neg a_2 \rangle$. *The 2-views De Morgan algebra is a special case of presenting inconsistent viewpoints ([24, 39]), when only 2 viewpoints are represented. That is, for* $\langle a_1, a_2 \rangle \in \{T, F\} \times \{T, F\}$, a_1 *represents the first view and* a_2 *represents the second view (note that these are different views of the same properties). The two views can be contradictory. For example,* TF *denotes an element in* $\{T, F\} \times \{T, F\}$, *where the first view is* T *and the second is* F.

The 3×3 structure is isomorphic to $\mathcal{P}(\mathcal{B}(\mathcal{D}))$*—the CPDB induced by* $\mathcal{B}(\mathcal{D})$. *As such it also represents two views, as* $\mathcal{D}$, *which may be contradictory. For*

example, the element TF of $\mathcal{D}$ is also an element of 3×3 which corresponds to the (consistent) element $\langle TF, FT\rangle$ from $\mathcal{B}(\mathcal{D})$. However, such elements, which represent contradictory views, should not be confused with inconsistent elements in $\mathcal{B}(\mathcal{D})$ such as $\langle TT, TT\rangle$, which do not exist in 3×3. (i.e., in the CPDB induced by $\mathcal{B}(\mathcal{D})$). The consistent elements of $\mathcal{B}(\mathcal{D})$ are mapped into pairs over {T,F,?} in the 3×3 structure. E.g., $\langle TF, FF\rangle$ (which is a consistent element of $\mathcal{B}(\mathcal{D})$ since $\langle TF, FF\rangle \leq_{\mathbf{i}} \langle TF, \neg TF\rangle$) is represented by T? and $\langle TT, FF\rangle$ is represented by TT. Intuitively, each pair in the 3×3 structure represents two different views, where now each of the views may be indefinite (?). The resulting structure, 3×3, 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 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 provide a different way of 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. The first set in an element of $\langle 2^{\{a,b\}}, 2^{\{a,b\}}\rangle$ consists of the views that are true, and the second consists of the views that are false. Thus both elements $\langle\{a\},\{b\}\rangle$ and TF represent the case of true by the first view and false by the second view. More generally this means that we may represent an element of the 3×3 structure as a vector of 3-valued attributes (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 indefinite elements are all the rest). The use of a specific structure (or representation) will depend, for example in this case, on the implementation.

Multi-valued Kripke models as well as the semantics of L_μ formulas and mc-graphs were defined over a De Morgan algebra $\mathcal{D}$. These definitions are easily extended to consider a *multi-valued structure* $\mathcal{B}$ which is either a distributive bilattice or a CPDB, and consists of both an information and a truth lattice. In this case, $\mathcal{B}_t$, the truth lattice of $\mathcal{B}$, functions as the De Morgan algebra $\mathcal{D}$ used in the original definitions provided in Section 2.

From now on, we also consider multi-valued Kripke models defined over a bilattice $\mathcal{B} = (B, \leq_{\mathbf{i}}, \leq_{\mathbf{t}}, \neg)$ which is either a distributive bilattice or a CPDB. Formally, a multi-valued Kripke model over $\mathcal{B}$ is a 6-tuple $M = \langle \mathcal{B}_t, AP, S, s_0, R, \Theta\rangle$, where AP, S, s_0, R, Θ are defined as in Definition 2.3, and $\mathcal{B}_t$ is the truth lattice of $\mathcal{B}$. Note that even a multi-valued model which is defined over a De Morgan algebra $\mathcal{D}$ whose lattice is not (isomorphic to) a truth lattice of a bilattice (or a CPDB) $\mathcal{B}$, can be viewed as a model over a bilattice since $\mathcal{D}$ can be lifted into $\mathcal{B}(\mathcal{D})$, resulting in a bilattice which contains both the original elements of $\mathcal{D}$ and their approximations. The model over $\mathcal{D}$ can then be interpreted as a model over $\mathcal{B}(\mathcal{D})$ or its CPDB, where each value of $\mathcal{D}$ 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 Section 7.

4. Mixed Simulation and Refinement of Multi-Valued Models

In this section 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 4.3, 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 (defined in Section 2.1). The simulation relation is defined by means of the information order of the multi-valued structure (Definition 3.1). Intuitively, the simulation relation 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* $M_1 = \langle \mathcal{B}_t, AP, S_1, s_0^1, R_1, \Theta_1 \rangle$ *and* $M_2 = \langle \mathcal{B}_t, AP, S_2, s_0^2, R_2, \Theta_2 \rangle$ *be two multi-valued Kripke models over* $\mathcal{B}= (B, \leq_{\mathbf{i}}, \leq_{\mathbf{t}}, \neg)$ *(where* $\mathcal{B}_t$ *is the truth lattice of* $\mathcal{B}$*).* $H \subseteq S_1 \times S_2$ *is a* mixed simulation *from* M_1 *to* M_2 *if* $(s_1, s_2) \in H$ *implies:*

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

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 $R_2(s_2, t_2) \not\leq_{\mathbf{i}} false$ have a representation in M_1. These requirements are similar to the requirements of mixed simulation in the 3-valued case ([29, 20]). There, every *may* transition in M_1 (i.e., a transition with value ?) has a representation in M_2, and every *must* transition in M_2 (i.e., a transition with value T) has a representation in M_1. In the multi-valued case, the equivalent of transitions which are *may* and not *must* are transitions for which $R(s, t) \leq_{\mathbf{i}} false$.

Example 4.2. *Consider the models in Figure 2(a) on page 25 and Figure 2(b) on page 25. The underlying multi-valued structure is the* 3×3 *structure (described in Figure 1(e),(f) on page 6). The mixed simulation is given by:* $H = \{(s_{00}, s_0), (s_{01}, s_0), (s_{10}, s_1), (s_{11}, s_1), (s_{20}, s_2), (s_{30}, s_3), (s_{31}, s_3)\}$*, and we can then 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* $F? \leq_{\mathbf{i}} FF$*.* $R_A(s_3, s_1) = F?$*, thus by (3) there is no requirement for a matching transition.*

Recall that in Section 2 we defined the interpretation $\|\varphi\|^M$ of $\varphi \in L_\mu$ in a multi-valued Kripke model M. The following theorem describes the correlation between mixed simulation relation and the interpretation of formulas in L_μ. Namely, it shows that the mixed simulation relation guarantees preservation of L_μ formulas.

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* $(s_1, s_2) \in H$*,* $\|\varphi\|^{M_2}(s_2) \leq_{\mathbf{i}} \|\varphi\|^{M_1}(s_1)$.

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 $\eta \in \{\mu, \nu\}$. According to Theorem 2.6, for every state $t_1 \in S_1$ there exists $\alpha(t_1) \in \mathbb{N}$ such that $\|\varphi'\|^{M_1}(t_1) = Z^{\alpha(t_1)}(t_1)$. Similarly for every $t_2 \in S_2$ there exists $\alpha(t_2) \in \mathbb{N}$ such that $\|\varphi'\|^{M_2}(t_2) = Z^{\alpha(t_2)}(t_2)$. Let $\alpha_1 = max\{\alpha(t_1) \mid t_1 \in S_1\}$ and $\alpha_2 = max\{\alpha(t_2) \mid t_2 \in S_2\}$. Note that α_1 and α_2 are well defined since S_1 and S_2 are finite. Finally, let $\alpha^* = max(\alpha_1, \alpha_2)$. Then for every $t_1 \in S_1$, $\|\varphi'\|^{M_1}(t_1) = Z^{\alpha^*}(t_1)$ and for every $t_2 \in S_2$, $\|\varphi'\|^{M_2}(t_2) = Z^{\alpha^*}(t_2)$. Recall that Z^{α^*} can also be defined as a 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 $\|\varphi\|^{M_2}(t_2) = \|\varphi'\|^{M_2}(t_2)$ for every $t_1 \in S_1$ and $t_2 \in S_2$. Furthermore, since φ is a closed L_μ, then φ' does not include any variables, nor does it include fixpoint subformulas.

We will prove the theorem on the transformed φ', by induction on the structure of the formula (which is fixpoint-free).

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: $\Theta_2(p)(s_2) \leq_{\mathbf{i}} \Theta_1(p)(s_1)$. Thus, $\|p\|^{M_2}(s_2) \leq_{\mathbf{i}} \|p\|^{M_1}(s_1)$.

Step:

- $\varphi = \neg p$ where $p \in AP$:

By definition of the multi-valued semantics (Section 2.1), $\|\neg p\|^{M_2}(s_2) = \neg\Theta_2(p)(s_2)$ and $\|\neg p\|^{M_1}(s_1) = \neg\Theta_1(p)(s_1)$.

By definition of H as mixed simulation, $\Theta_2(p)(s_2) \leq_{\mathbf{i}} \Theta_1(p)(s_1)$. Since $\neg$ is monotone with respect to $\leq_{\mathbf{i}}$ (by Definition 3.1), we can conclude that $\|\neg p\|^{M_2}(s_2) \leq_{\mathbf{i}} \|\neg p\|^{M_1}(s_1)$.

- $\varphi = \varphi_1 \wedge \varphi_2$:

  By definition of the multi-valued semantics, $\|\varphi_1 \wedge \varphi_2\|^{M_i}(s_i) = \|\varphi_1\|^{M_i}(s_i) \wedge \|\varphi_2\|^{M_i}(s_i)$ for $i \in \{1, 2\}$.

  By the induction assumption, $\|\varphi_1\|^{M_2}(s_2) \leq_{\mathbf{i}} \|\varphi_1\|^{M_1}(s_1)$ and $\|\varphi_2\|^{M_2}(s_2) \leq_{\mathbf{i}} \|\varphi_2\|^{M_1}(s_1)$.

  Since $\wedge$ is monotone with respect to $\leq_{\mathbf{i}}$ (by Definition 3.1), we can conclude that: $\|\varphi_1\|^{M_2}(s_2) \wedge \|\varphi_2\|^{M_2}(s_2) \leq_{\mathbf{i}} \|\varphi_1\|^{M_1}(s_1) \wedge \|\varphi_2\|^{M_1}(s_1)$.

  Thus, $\|\varphi_1 \wedge \varphi_2\|^{M_2}(s_2) \leq_{\mathbf{i}} \|\varphi_1 \wedge \varphi_2\|^{M_1}(s_1)$.

- $\varphi = \varphi_1 \vee \varphi_2$: Dual to the case of $\varphi = \varphi_1 \wedge \varphi_2$.

- $\varphi = \Box\varphi_1$:

  By definition of the multi-valued semantics, $\|\Box\varphi_1\|^{M_i}(s_i) = \bigwedge_{t_i \in S_i} \big(\neg R_i(s_i, t_i) \vee \|\varphi_1\|^{M_i}(t_i)\big)$ for $i \in \{1, 2\}$.

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

  - $\forall j \exists k. a_j \leq_{\mathbf{i}} b_k$ or $a_j \leq_{\mathbf{i}} \mathit{true}$ and
  - $\forall k \exists j. a_j \leq_{\mathbf{i}} b_k$

  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 $\bigwedge_{t_2 \in S_2} \big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \bigwedge_{t_1 \in S_1} \big(\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1)\big)$.

  We first prove that for all $t_2 \in S_2$, either $\big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \mathit{true}$ or there exists $t_1 \in S_1$ such that $\big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \big(\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1)\big)$.

  We then prove that for all $t_1 \in S_1$ there exists $t_2 \in S_2$ such that $\big(\neg R_2(s_2, t_2) \vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \big(\neg R_1(s_1, t_1) \vee \|\varphi_1\|^{M_1}(t_1)\big)$.

  For each $t_2 \in S_2$:
  If $R_2(s_2, t_2) \not\leq_{\mathbf{i}} \mathit{false}$, then by the definition of mixed simulation, there exists $t_1 \in S_1$ such that $(t_1, t_2) \in H$ and $R_2(s_2, t_2) \leq_{\mathbf{i}} R_1(s_1, t_1)$. As negation is monotone with respect to $\leq_{\mathbf{i}}$, we have: $\neg R_2(s_2, t_2) \leq_{\mathbf{i}} \neg R_1(s_1, t_1)$. Since $(t_1, t_2) \in H$, then by induction assumption: $\|\varphi_1\|^{M_2}(t_2) \leq_{\mathbf{i}} \|\varphi_1\|^{M_1}(t_1)$.

Since $\vee$ is monotone with respect to $\leq_{\mathbf{i}}$, then $\big(\neg R_2(s_2,t_2)\vee\|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \big(\neg R_1(s_1,t_1)\vee\|\varphi_1\|^{M_1}(t_1)\big)$.

For the case where $R_2(s_2,t_2) \leq_{\mathbf{i}} false$, since negation is monotone with respect to $\leq_{\mathbf{i}}$, then $\neg R_2(s_2,t_2) \leq_{\mathbf{i}} true$.
Since $\vee$ is monotone with respect to $\leq_{\mathbf{i}}$, we can conclude that: $\big(\neg R_2(s_2,t_2)\vee \|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \big(true \vee \|\varphi_1\|^{M_2}(t_2)\big) = true$

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 $\big(\neg R_2(s_2,t_2)\vee\|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \big(\neg R_1(s_1,t_1)\vee\|\varphi_1\|^{M_1}(t_1)\big)$.
- $\big(\neg R_2(s_2,t_2)\vee\|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} true$

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 $R_2(s_2,t_2) \leq_{\mathbf{i}} R_1(s_1,t_1)$. Thus, again we can conclude that: $\big(\neg R_2(s_2,t_2)\vee\|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \big(\neg R_1(s_1,t_1)\vee\|\varphi_1\|^{M_1}(t_1)\big)$.

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: $\bigwedge_{t_2\in S_2}\big(\neg R_2(s_2,t_2)\vee\|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \bigwedge_{t_1\in S_1}\big(\neg R_1(s_1,t_1)\vee\|\varphi_1\|^{M_1}(t_1)\big)$.

We conclude, $\|\Box\varphi_1\|^{M_2}(s_2) \leq_{\mathbf{i}} \|\Box\varphi_1\|^{M_2}(s_1)$.

- $\varphi = \Diamond\varphi_1$:

  By definition of the multi-valued semantics, $\|\Diamond\varphi_1\|^{M_i}(s_i) = \bigvee_{t_i\in S_i}\big(R(s_i,t_i)\wedge\|\varphi_1\|^{M_i}(t_i)\big)$ for $i \in \{1,2\}$.

  We first notice that proving $\bigvee_{j=1}^{n} a_j \leq_{\mathbf{i}} \bigvee_{j=1}^{m} b_j$ is similar to proving $\bigvee_{j=1}^{n} a_j \leq_{\mathbf{i}} \bigvee_{j=1}^{m} b_j \vee false$. For proving that, since $\vee$ is monotone with respect to $\leq_{\mathbf{i}}$, it suffices to show that:

  - $\forall j \exists k. a_j \leq_{\mathbf{i}} b_k$ or $a_j \leq_{\mathbf{i}} false$ and
  - $\forall k \exists j. a_j \leq_{\mathbf{i}} b_k$

  We need to prove $\bigvee_{t_2\in S_2}\big(R_2(s_2,t_2)\wedge\|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \bigvee_{t_1\in S_1}\big(R_1(s_1,t_1)\wedge\|\varphi_1\|^{M_1}(t_1)\big)$.

  We first prove that for all $t_2 \in S_2$, either $\big(R_2(s_2,t_2)\wedge\|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} false$ or there exists $t_1 \in S_1$ such that $\big(R_2(s_2,t_2)\wedge\|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \big(R_1(s_1,t_1)\wedge\|\varphi_1\|^{M_1}(t_1)\big)$.

  We then prove that for all $t_1 \in S_1$ there exists $t_2 \in S_2$ such that $\big(R_2(s_2,t_2)\wedge\|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \big(R_1(s_1,t_1)\wedge\|\varphi_1\|^{M_1}(t_1)\big)$.

For each $t_2 \in S_2$:
If $R_2(s_2, t_2) \not\leq_{\mathbf{i}} false$, then by the definition of mixed simulation, there exists $t_1 \in S_1$ such that $(t_1, t_2) \in H$ and $R_2(s_2, t_2) \leq_{\mathbf{i}} R_1(s_1, t_1)$.
Since $(t_1, t_2) \in H$, then by induction assumption: $\|\varphi_1\|^{M_2}(t_2) \leq_{\mathbf{i}} \|\varphi_1\|^{M_1}(t_1)$.
Since $\wedge$ is monotone with respect to $\leq_{\mathbf{i}}$, we conclude that $R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2) \leq_{\mathbf{i}} R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)$.

For the case where $R_2(s_2, t_2) \leq_{\mathbf{i}} false$. Since $\wedge$ is monotone with respect to $\leq_{\mathbf{i}}$, we can conclude that: $\big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \big(false \wedge \|\varphi_1\|^{M_2}(t_2)\big) = false$

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 $\big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \big(R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)\big)$.
- $\big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} false$

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 $R_2(s_2, t_2) \leq_{\mathbf{i}} R_1(s_1, t_1)$. Thus, again we can conclude that: $\big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \big(R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)\big)$.

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: $\bigvee_{t_2 \in S_2} \big(R_2(s_2, t_2) \wedge \|\varphi_1\|^{M_2}(t_2)\big) \leq_{\mathbf{i}} \bigvee_{t_1 \in S_1} \big(R_1(s_1, t_1) \wedge \|\varphi_1\|^{M_1}(t_1)\big)$.

We conclude, $\|\Diamond\varphi_1\|^{M_2}(s_2) \leq_{\mathbf{i}} \|\Diamond\varphi_1\|^{M_1}(s_1)$.

□

Intuitively, Theorem 4.3 implies that for multi-valued models that are related by a mixed simulation relation, $M_1 \preceq M_2$, the model checking result over M_1 is at least as informative as the model checking result over M_2. This justifies the use of abstraction based on mixed simulation as a means to evaluate μ-calculus formulas in a sound manner.

4.2. Concrete and Abstract Multi-Valued Models

Abstraction typically 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 zero 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. When multi-valued models are considered, abstraction manifests itself also in the multi-valued structure over which the model is defined.

A model is more abstract than another model if the latter model has equally or more information than the former one.

In our setting, a concrete model, denoted M_C, is a multi-valued model over a De Morgan algebra $\mathcal{D}= (D, \leq, \neg)$. Recall that, as discussed in Section 3.1, for any element $c \in D$, we have that $\langle x, y\rangle \in D \times D$ approximates c if $\langle x, y\rangle \leq_{\mathbf{i}} \langle c, \neg c\rangle$ (where $\leq_{\mathbf{i}}$ is the information order of the bilattice induced by $\mathcal{D}$, as described in Definition 3.2). Since $\langle c, \neg c\rangle$ is consistent, and all $\langle x, y\rangle$ where $\langle x, y\rangle \leq_{\mathbf{i}} \langle c, \neg c\rangle$ 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 M_C defined over $\mathcal{D}$, a model M_A which is an *abstract model* of M_C is defined over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$. We will not discuss how abstract models are constructed. This is investigated for instance in [38].

The mixed simulation relation can be used to describe the relation between a concrete model, M_C, defined over $\mathcal{D}$ and its abstraction, M_A defined over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$: $M_C \preceq M_A$. 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 the mixed simulation relation was defined for models that are both defined over the same multi-valued structure, whereas in our case M_C is defined over $\mathcal{D}$ and M_A is defined over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$. To overcome this problem, we interpret M_C as a model over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$, instead of over $\mathcal{D}$, in which all values are definite. Following Theorem 3.11, evaluating M_C 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 concrete model M_C 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.

Note that the concrete model M_C is in itself multi-valued (and hence possibly abstract). We refer to it as concrete since all its values are definite, and not approximated.

Example 4.4. *Consider the model M_C in Figure 2(a) on page 25. The underlying multi-valued structure is the 3×3 structure (described in Figure 1(e),(f) on page 6). This is a concrete model, as all its values are definite. Figure 2(b) describes an abstraction of the model such that $M_C \preceq M_A$ (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 related to s_A by H. 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 (neither from s_{30} nor from s_{31}). 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. This transition might still be created during the construction of the abstract model due to abstraction which is not precise.

4.3. Refinement Algorithm For Multi-Valued 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 whose information level is not maximal, and hence it does not provide the most 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 most information possible.

Let M_C be a concrete model over $\mathcal{D}$, and let M_A be an abstract model over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$, i.e. $M_C \preceq M_A$, such that M_A should be refined (i.e., the model checking result over M_A is indefinite). Our refinement consists of two parts:

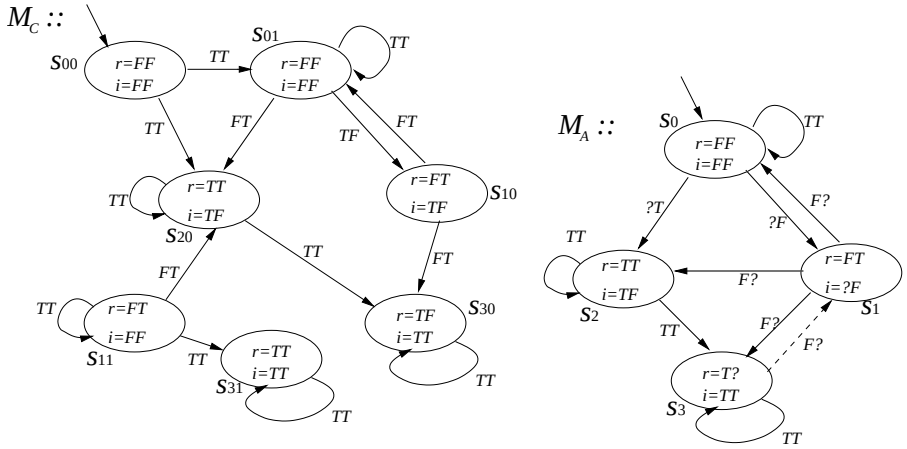
- First, we choose a *criterion* for model refinement.
- Then, based on the criterion, the model is refined by either increasing the information level of some transition or of an atomic proposition in some state, or by splitting states.

These refinement steps are similar to the refinement steps in [40]. There, refinement is done by either splitting an abstract state (as here), or by removing a transition. The latter results in a more precise model, M'_A, over the same state space. In our general multi-valued setting, this is similar to a refinement step which increases the information level of a transition or an atomic proposition in a multi-valued model.

Splitting states is also aimed at increasing the information level of some transition or of an atomic proposition in some state. Note, however, that splitting states can, as a by-product, cause a decrease in the information level of some transitions. If the refinement only increases information level, then $M_C \preceq M'_A \preceq M_A$.

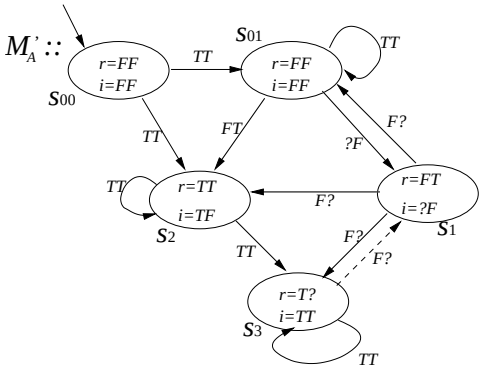
Example 4.5. *Figure 2(c) on page 25 describes a refinement of the abstract model M_A given in Figure 2(b) on page 25. The refinement is done by state splitting: s_0 is split into s_{00} and s_{01}. Splitting these states enables to increase the information level of the transition from s_0 to s_2. The 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. The described refinement only increases information level, thus $M_C \preceq M'_A \preceq M_A$.*

A different refinement of the abstract model M_A is given in Figure 2(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

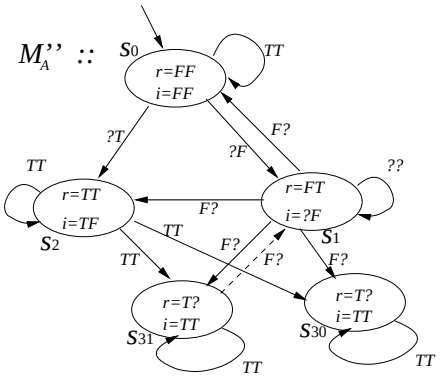


(a) Concrete Model

(b) Abstract Model



(c) Refined Abstract Model



(d) Another Refined Abstract Model

Figure 2: A concrete model M_C and its abstractions M_A, M'_A and M''_A

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.

The refinement is applied directly on the mc-graph. In fact, it suffices to refine the indefinite subgraph, where the mc-graph is pruned in definite nodes.

4.3.1. Finding a 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. For an mc-function $\chi : N \to L$, given 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 in n^0. The criterion for the refinement is obtained from a failure node. This is a node $n = s \vdash \varphi \in N$ such that (1) $\chi(n)$ is indefinite; (2) $\chi(n)$ affects $\chi(n^0)$; and (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.

Requirement (3) means that n *itself* is responsible for introducing (some) uncertainty. (1) and (2) require that this uncertainty is relevant to $\chi(n^0)$.

We adapt the **mc-algorithm** (Algorithm 2.8) 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 for refining the model. From now on we refer to the failure node and reason as *failure data*.

For every terminal node n, if $\chi(n)$ is indefinite, the failure node and reason attached to it are both the node itself. To handle nonterminal nodes, we define an auxiliary function $f : N \to L$. Recall that for nodes in a non-trivial MSCC, the algorithm iteratively updates a temporary value $temp(n)$ before it sets their final value $\chi(n)$. The auxiliary function f 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 $f(n) = temp(n)$.

We can now describe the algorithm **find-failure** for obtaining failure data for non-terminal nodes, which is invoked whenever $f(n)$ is updated:

Algorithm 4.6 (find-failure). *Let n be a node for which $f(n)$ has been updated last. If $f(n)$ is definite, then no failure data is 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 $n' \neq n''$ and $R(n,n'') \neq false$, $(R(n,n'') \wedge f(n'')) \leq_{\mathbf{t}} (R(n,n') \wedge f(n'))$ or $(R(n,n'') \wedge f(n''))$ and $(R(n,n') \wedge f(n'))$ are uncomparable.*
   (b) *$R(n,n') \wedge f(n')$ is indefinite.*

   Requirement (a) means that $R(n,n') \wedge f(n')$ is maximal. For the given node n and a chosen node n' satisfying (a),(b) define failure data for n as follows:

   i If $f(n')$ is definite or $R(n,n') \lneq_{\mathbf{t}} f(n')$: n is a failure node, and the edge (n,n') is the failure reason.

ii *If $R(n,n')$ is definite or $f(n') \lesssim_{\mathbf{t}} R(n,n')$, then the failure data of n is this of n'.*

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

2. *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* $n' \neq n''$ *and* $R(n,n'') \neq false$, $(\neg R(n,n') \vee f(n')) \leq_{\mathbf{t}} (\neg R(n,n'') \vee f(n''))$ *or* $(\neg R(n,n') \vee f(n'))$ *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 failure data for n as follows:

   i *If $f(n')$ is definite or $f(n') \lesssim_{\mathbf{t}} \neg R(n,n')$, then n is a failure node, the edge (n,n') is the failure reason.*

   ii *If $R(n,n')$ is definite or $\neg R(n,n') \lesssim_{\mathbf{t}} f(n')$, then the failure data of n is this of n'.*

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

Intuitively, requirements (a) and (b) in cases 1 and 2 of the algorithm ensure that the son n' of n affects $f(n)$ and that refinement is applicable. For example, if requirement (a) holds for a son n' of a $\vee$-node n (case 1), then $R(n,n') \wedge f(n')$ is maximal w.r.t. $\leq_{\mathbf{t}}$, and thus affects $f(n)$. Requirement (b) ensures that it is possible to refine $R(n,n')$ or $f(n')$. The case where n is a $\wedge$-node (case 2) is dual, where instead of searching for a maximal $R(n,n') \wedge f(n')$, the algorithm tries to find n' such that $\neg R(n,n') \vee f(n')$ is minimal.

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 $R(n,n') \lesssim_{\mathbf{t}} f(n')$, 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 failure search is formalized by the following lemma.

Lemma 4.7. *For every node n, if $f(n)$ is given an indefinite value, then there exists n' such that the following conditions hold:*

- *$R(n,n') \neq false$ and n' satisfies requirements 1(a) and 1(b) of **find-failure** if n is a $\vee$-node, and requirements 2(a) and 2(b) if n is a $\wedge$-node,*
- *$f(n')$ is already defined at the time **find-failure** searches for n', and*
- *if the update of failure data of n is based on n', then n' already has failure data.*

Proof: We prove each part of the lemma separately.

First, we prove the second item. That is, we prove that if $f(n)$ is given an indefinite value and $R(n,n') \neq false$, 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 every node n' which is son of n.

Second, we prove the first item, i.e., 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) of case 1 or case 2. We prove the lemma according to the type of the node n whose color $f(n)$ is indefinite. We first consider the case of a $\vee$-node. Assume by way of contradiction that no such son exists. This means that for every son n', if requirement (a) holds then $(R(n,n') \wedge f(n'))$ is definite. Note that if $b \leq_{\mathbf{t}} a$ then $a \vee b = a$ (by the definition of $\vee$). This implies that only sons complying with requirement (a) influence $f(n)$. Thus, by Theorem 3.11, since all these values are definite, $f(n)$ is also definite, contradicting the assumption that its color is indefinite. The case of $\wedge$-nodes is dual. This means that if $f(n)$ is indefinite, then there exists n' which satisfies requirements(a),(b).

Third, we prove that if the updating of failure data on n is based on the failure data of n', then failure data is already defined on n'. By the description of the failure data update, if the failure data of n is based on the failure data 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 failure data. □

A failure data 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 data for n is based on χ. Also note that it is possible to keep at each step for some node n, only a failure son n' which satisfies requirements (a) and (b). Updating the failure data can be done after the **mc-algorithm** terminates, and thus can be calculated only once for each 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 translates to 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 does not guarantee that all possible refinements of the failure data will increase the information level of the result. It greedily searches for failure data which is most likely to increase the result

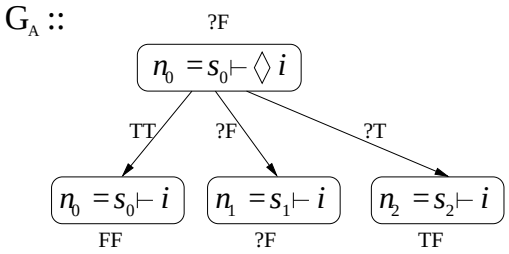


Figure 3: mc-graph for $\Diamond i$ on M_A

with respect to the information order. On the other hand, the algorithm will not return as failure data a node whose refinement will certainly not affect the result. For instance, subtrees in which the source node is definite are eliminated, and their nodes will not be returned as failure data.

Example 4.8. *Consider the mc-graph in Figure 3 on page 29. This is the mc-graph created for verifying the property $\Diamond i$ on the abstract model M_A in Figure 2(b) on page 25. The mc-graph is constructed as explained in Section 2.2.1. For the node n_0 marked $s_0 \vdash \Diamond i$ 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 marked $s_2 \vdash i$ being the reason. The second is node n_1 marked $s_1 \vdash i$ being the failure data, and the third is node n_0 itself and the edge to n_1 being the reason.*
Assume the failure data returned for model checking the model M_A (Figure 2(b)) is the node n_0 itself being the failure node and the edge to node n_2 being the reason. A refinement of M_A based on the given failure data results in the model M'_A (Figure 2(c) on page 25). Increasing the information level of the edge from s_0 to s_2 can only be done by splitting the state s_0 into two states.

Recall that refinement is done by either increasing the information level of some transition or atomic proposition, or by splitting states. If refinement is done by splitting states, then every abstract state of the model will represent less concrete states. If the underlying concrete model is finite, then there is a finite number of such refinements possible. Since the information lattice of the underlying multi-valued structure is finite, there is also a finite number of refinement steps which increase the information. We conclude that there is a finite number of refinement steps possible. This is formalized in the following lemma.

Lemma 4.9. *If M_C is finite then in a finite number of refinement steps starting from M_A a model M'_A will be obtained such that the model checking result over M'_A is equal to the model checking result over the underlying concrete model, M_C.*

Proof: At each refinement step we either increase the information level on some transition or on an atomic proposition in some state, or split states.

If refinement is done by state splitting, then 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.

Since the information lattice of the underlying multi-valued structure is finite, then there is a finite number of refinement steps which increase the information level on some transition or on an atomic proposition in some states. Note that even if AP is infinite, there is only a finite number of atomic propositions in the checked formula, φ, and refinement will increase information only on the atomic propositions in the formula.

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 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, $\|\varphi\|^{M'_A} \leq_{\mathbf{i}} \|\varphi\|^{M_C}$. 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.

□

5. Partial Coloring and Subgraphs

In this section we investigate properties of the **mc-algorithm** (Algorithm 2.8). In particular, we present sufficient conditions under which a subgraph of an mc-graph can be evaluated "correctly" (the notion "correct" will be formally defined later) without considering the full mc-graph. Given a subgraph of the mc-graph and a valuation for some of the subgraph's nodes, we show how to complete the evaluation of the subgraph in a correct manner.

The results presented in this section will assist us when presenting our compositional model checking framework in Section 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 section, G denotes a multi-valued mc-graph over a multi-valued structure $\mathcal{B}= (B, \leq_{\mathbf{t}}, \leq_{\mathbf{i}}, \neg)$. The set of nodes of G is denoted N.

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 an mc-graph and let $f : N \to B$ 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 *f with respect to n, if one of the following holds:*

- *n is a $\wedge$-node and*
  1. $(\neg R(n, n') \vee f(n')) \ \leq_{\mathbf{t}} \ (\neg R(n, n'') \vee f(n''))$, *and*

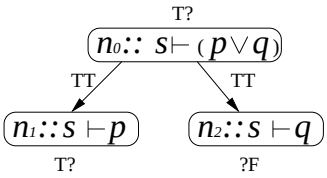


Figure 4: Influencing sons on mc-graph

2. *For every* $v', v'' \in B$: *if* $f(n') \leq_{\mathbf{i}} v'$ *and* $f(n'') \leq_{\mathbf{i}} v''$ *then* $(\neg R(n,n') \vee v') \leq_{\mathbf{t}} (\neg R(n,n'') \vee v'')$.

- n *is a* $\vee$*-node and*
  1. $(R(n,n'') \wedge f(n'')) \leq_{\mathbf{t}} (R(n,n') \wedge f(n'))$
  2. *For every* $v', v'' \in B$: *if* $f(n') \leq_{\mathbf{i}} v'$ *and* $f(n'') \leq_{\mathbf{i}} v''$ *then* $(R(n,n'') \wedge v'') \leq_{\mathbf{t}} (R(n,n') \wedge v')$.

Intuitively, a son n' covers a son n'' in the sense that if f 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, f 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 for every $f' \geq_{\mathbf{i}} f$ as well. 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 4 on page 31. Assume the underlying structure is the* 3×3 *structure (Figure 1(e), (f) on page 6). By the definition of the mc-graph, the value of the edges are* $R(n_0,n_1) = R(n_0,n_2) = TT$. *Let* $f(n_1) = T?$ *and* $f(n_2) = ?F$. *We show that* n_2 *is covered by* n_1 *under* f. *Clearly, requirement (1) holds. For requirement (2), we need to show that* $\forall v_1, v_2 \in B$: *if* $f(n_1) \leq_{\mathbf{i}} v_1$ *and* $f(n_2) \leq_{\mathbf{i}} v_2$ *then* $(R(n_0,n_2) \wedge v_2) \leq_{\mathbf{t}} (R(n_0,n_1) \wedge v_1)$. *Specifically, we need to show that* $\forall v_1 \in \{T?, TT, TF\}$, $\forall v_2 \in \{?F, FF, TF\}$: $TT \wedge v_2 \leq_{\mathbf{t}} TT \wedge v_1$, *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.

Definition 5.3. *Let* G *be an 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 an mc-graph,* χ *its mc-function, and* $\chi_I : N \to B$ *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. Resulting from that, 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 an 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 mc-function (replacing val) results in an 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 valuation of G' will be the same as the valuation of G. For the terminal nodes of G' this is clear, since the initial value is defined over all terminal nodes and is correct 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: $\chi(n) = \bigvee\{R(n,n') \wedge \chi(n') | R(n,n') \neq false\}$ (Algorithm 2.8). 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 $\big(R(n,n'') \wedge \chi(n'')\big) \leq_{\mathbf{t}} \big(R(n,n') \wedge \chi(n')\big)$. $a \leq_{\mathbf{t}} b$ 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.

□

6. Compositional Model Checking

In this section 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 the partial model, 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 model separately will help us in model checking the composed system, without fully constructing it. Based on the results from Section 5, we will be able to deduce about the full system from a partially constructed composed system.

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.

For the rest of the section we fix a composed system $M_1||M_2$, where M_1 and M_2 are multi-valued Kripke models defined over the same CPDB $\mathcal{P}(\mathcal{B}(\mathcal{D}))= (B, \leq_{\mathbf{t}}, \leq_{\mathbf{i}}, \neg)$. In the following, for $i \in \{1, 2\}$ we assume that every (abstract) model M_i has an underlying concrete model M_i^c defined over $\mathcal{D}$, such that $M_i^c \preceq M_i$. Thus, if the model M_i is abstract (i.e., contains indefinite values), it can be refined with respect to its underlying concrete model, M_i^c. Let $M_i = \langle \mathcal{B}_t, AP_i, S_i, s_0^i, R_i, \Theta_i \rangle$, where $\mathcal{B}_t$ is the truth lattice of $\mathcal{P}(\mathcal{B}(\mathcal{D}))$, we use $\bar{i}$ to denote the remaining index in $\{1, 2\} \setminus \{i\}$.

6.1. Composition of Models

We start by defining the composition $M_1||M_2$. For that we need the following definitions.

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 $\mathcal{P}(\mathcal{B}(\mathcal{D}))$ is a CPDB, and in particular a partial distributive bilattice (Definition 3.6). 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 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 5(a) on page 36. The underlying multi-valued structure is the 3×3 structure (described in Figure 1(e),(f) on page 6). 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 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 $M = \langle \mathcal{B}_t, AP, S, s_0, R, \Theta \rangle$ over $\mathcal{P}(\mathcal{B}(\mathcal{D}))$, 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 5(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 might 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 in Definition 6.4 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 $\bot$ 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 $\bot$. This ensures that the value of the composed transition is not "more true" than $\bot$, but may be "more false" than $\bot$, and thus more informative. More precisely, consider the CPDB $\mathcal{P}(\mathcal{B}(\mathcal{D}))$. Then $\bot$ in $\mathcal{P}(\mathcal{B}(\mathcal{D}))$ is defined as $\langle d_\bot, d_\bot \rangle \in D \times D$ where for every a in D, $d_\bot \leq a$. Thus, for every $\langle a, b \rangle \in D \times D$, $\langle a, b \rangle \wedge \bot = \langle d_\bot, b \rangle$, 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 [58], 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 5(a) on page 36. 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 5(b) on page 36. 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, although, 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 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.*

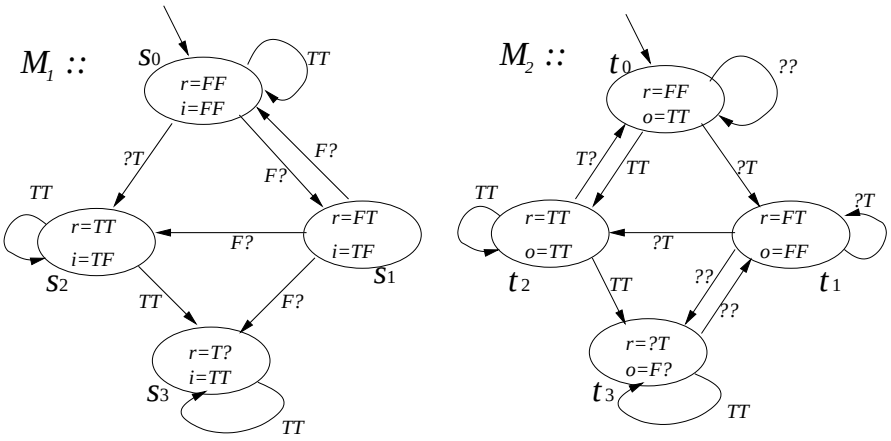
6.2. Lifting Models

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 $M_1 || M_2$.

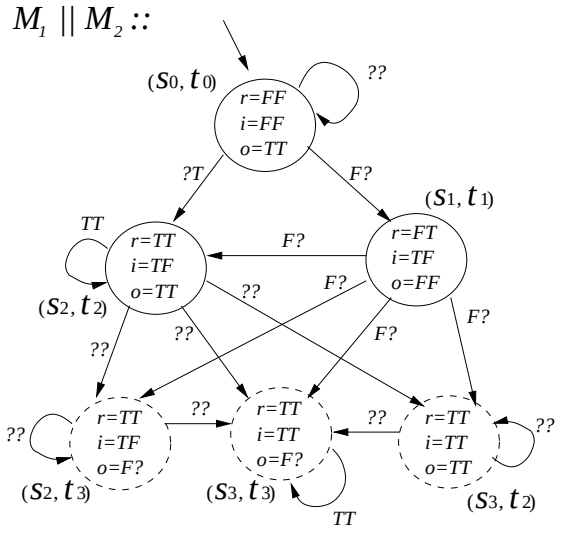
From now on we fix AP to be $AP_1 \cup AP_2$.

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

- *For each $s_i, t_i \in S_i$: $R_i \uparrow (s_i, t_i) = R_i(s_i, t_i) \wedge \bot$.*



(a) Components M_1 and M_2



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

Figure 5: Components M_1, M_2 and their composition

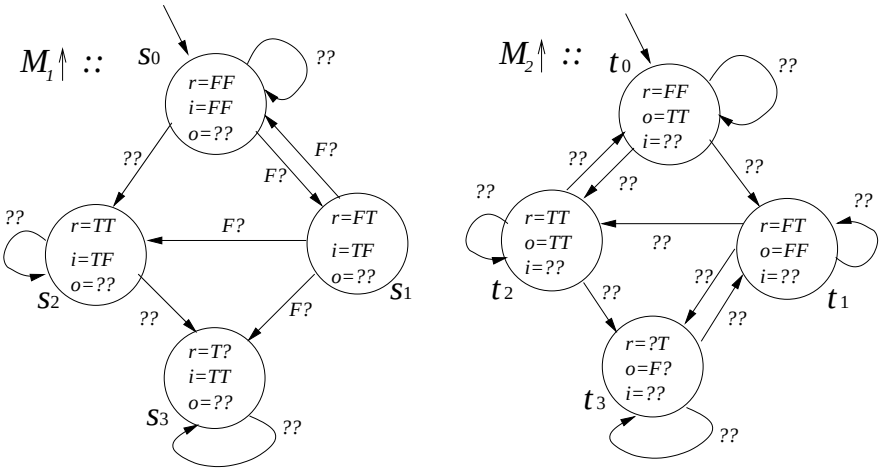


Figure 6: Lifted models for M_1 and M_2

- *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 $\Theta_i\uparrow(p)(s_i) = \bot$.

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 ($\bot$). 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 $\bot$. 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 5(a) on page 36 are presented in Figure 6 on page 37. 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.

Theorem 6.9. *For every $i \in \{1,2\}$, $M_1||M_2 \preceq M_i\uparrow$. The mixed simulation relation $H \subseteq S \times S_i$ is given by $\{((s_1,s_2),s_i)|(s_1,s_2) \in S\}$*

Proof: Let $H \subseteq S \times S_i$ be the relation defined by $\{((s_1,s_2),s_i)|(s_1,s_2) \in S\}$. That is, a state, $s_i \in M_i\uparrow$ is related by H to all the states of $M_1\|M_2$ where the corresponding state in the pair is s_i.

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 $(s,s_i) \in H$. Then:

1. Prove that $\forall p \in AP : \Theta(p)(s) \leq_{\mathbf{i}} \Theta_i{\uparrow}(p)(s_i)$
   - $p \in AP_i \setminus AP_{\bar{i}}$: By the definition of the composition: $\Theta(p)(s) = \Theta_i(p)(s_i)$. By the definition of $M_i{\uparrow}$, $\Theta_i{\uparrow}(p)(s_i) = \Theta_i(p)(s_i)$. Thus, $\Theta(p)(s) = \Theta_i \uparrow (p)(s_i)$, which immediately implies $\Theta_i \uparrow (p)(s_i) \leq_{\mathbf{i}} \Theta(p)(s)$.
   - $p \in AP_1 \cap AP_2$: By the definition of the composition: $\Theta(p)(s) = \Theta_1(p)(s_1) \oplus \Theta_2(p)(s_2)$. Recall that $\oplus$ is the join operator on the information lattice (Definition 3.1), thus for every elements a, b in the bilattice, $a \leq_{\mathbf{i}} a \oplus b$. Therefore $\Theta_i(p)(s_i) \leq_{\mathbf{i}} \Theta_1(p)(s_1) \oplus \Theta_2(p)(s_2)$. By definition of $M_i{\uparrow}$, $\Theta_i{\uparrow}(p)(s_i) = \Theta_i(p)(s_i)$. Thus, $\Theta_i{\uparrow}(p)(s_i) \leq_{\mathbf{i}} \Theta(p)(s)$.
   - $p \in AP_{\bar{i}} \setminus AP_i$: By the definition of the composition: $\Theta(p)(s) = \Theta_{\bar{i}}(p)(s_{\bar{i}})$. By the definition of $M_i \uparrow$: $\Theta_i \uparrow (p)(s_i) = \bot$. Thus, $\Theta_i{\uparrow}(p)(s_i) \leq_{\mathbf{i}} \Theta(p)(s)$.
2. Prove that for every $t \in S$ such that $R(s,t) \neq false$ there exists $t_i \in S_i$ such that $(t, t_i) \in H$ and $R_i{\uparrow}(s_i, t_i) \leq_{\mathbf{i}} R(s,t)$.
   Let $t = (t_1, t_2) \in S$ and let $R(s,t) \neq false$.
   Since $R(s,t) \neq false$, we have $R_i(s_i, t_i) \neq false$. By the definition of $M_i{\uparrow}$, $R_i{\uparrow}(s_i, t_i) = R_i(s_i, t_i) \wedge \bot$. By definition of the composed model, $R(s,t) = R_i(s_i, t_i) \wedge R_{\bar{i}}(s_{\bar{i}}, t_{\bar{i}})$ (if t_1 and t_2 are composable) or $R_i(s_i, t_i) \wedge R_{\bar{i}}(s_{\bar{i}}, t_{\bar{i}}) \wedge \bot$ (if t_1 and t_2 are weakly composable and not composable). Since $\bot \leq_{\mathbf{i}} R_{\bar{i}}(s_{\bar{i}}, t_{\bar{i}})$, and since $\wedge$ is monotone with respect to $\leq_{\mathbf{i}}$ (Definition 3.1), we conclude that $R_i{\uparrow}(s_i, t_i) \leq_{\mathbf{i}} R(s,t)$. Now pick $t'_i = t_i$. $(t, t'_i) \in H$ and $R_i{\uparrow}(s_i, t'_i) \leq_{\mathbf{i}} R(s,t)$. Thus t'_i satisfies the proof requirement.
3. Prove that for every $t_i \in S_i$ such that $R_i \uparrow (s_i, t_i) \not\leq_{\mathbf{i}} false$ there exists $t \in S$ such that $(t, t_i) \in H$ and $R_i{\uparrow}(s_i, t_i) \leq_{\mathbf{i}} R(s,t)$.
   Let $t_i \in S_i$ and $R_i{\uparrow}(s_i, t_i) \not\leq_{\mathbf{i}} false$.
   By definition of $M_i{\uparrow}$, the value of $R_i{\uparrow}(s_i, t_i) = R_i(s_i, t_i) \wedge \bot$. $\bot \leq_{\mathbf{i}} false$ and $R_i(s_i, t_i) \leq_{\mathbf{i}} R_i(s_i, t_i)$. Since $\wedge$ is monotone with respect to $\leq_{\mathbf{i}}$, we conclude that $R_i{\uparrow}(s_i, t_i) = R_i(s_i, t_i) \wedge \bot \leq_{\mathbf{i}} R_i(s_i, t_i) \wedge false = false$. Thus, the proof requirement holds trivially, there is no t_i such that $R_i{\uparrow}(s_i, t_i) \not\leq_{\mathbf{i}} false$ in $M_i{\uparrow}$.

□

Example 6.10. *Based on Theorem 6.9, we conclude that the lifted model $M_1{\uparrow}$ (Figure 6 on page 37) is an abstraction of the composed system $M_1||M_2$ (Figure 5(b) on page 36), thus $M_1||M_2 \preceq M_1{\uparrow}$. 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,

$\|\varphi\|^{M_i\uparrow} \leq_{\mathbf{i}} \|\varphi\|^{M_1||M_2}$. 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 on page 37), and the composed model $M_1||M_2$ (Figure 5(b) on page 36).*

Since the value of i in s_2 is TF, and the value of o in it is ??, we get that $\|\varphi\|^{M_1\uparrow}(s_2) = T?$. Similarly, $\|\varphi\|^{M_2\uparrow}(t_2) = ?T$. 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 $\|\varphi\|^{M_1\uparrow}(s_2) \leq_{\mathbf{i}} \|\varphi\|^{M_1||M_2}((s_2,t_2))$, thus $T? \leq_{\mathbf{i}} \|\varphi\|^{M_1||M_2}((s_2,t_2))$. Similarly, $\|\varphi\|^{M_2\uparrow}(t_2) \leq_{\mathbf{i}} \|\varphi\|^{M_1||M_2}((s_2,t_2))$, i.e. $?T \leq_{\mathbf{i}} \|\varphi\|^{M_1||M_2}((s_2,t_2))$. However, this still does not result in a definite value for $\|\varphi\|^{M_1||M_2}((s_2,t_2))$.

Still, since $\|\varphi\|^{M_1\uparrow}(s_2) \oplus \|\varphi\|^{M_2\uparrow}(t_2) \leq_{\mathbf{i}} \|\varphi\|^{M_1||M_2}((s_2,t_2))$, we conclude: $T?\oplus?T = TT \leq_{\mathbf{i}} \|\varphi\|^{M_1||M_2}((s_2,t_2))$. Thus, $\|\varphi\|^{M_1||M_2}((s_2,t_2)) = TT$, which is a definite value.

This example demonstrates how a definite model checking result for $M_1||M_2$ is obtained by combining two indefinite results from $M_1\uparrow$ and $M_2\uparrow$.

A more typical case is when $\|\varphi\|^{M_1\uparrow} \oplus \|\varphi\|^{M_2\uparrow}$ is still indefinite. This reflects the fact that φ not only depends on both components, but it also depends on their synchronization. Typically, such a result requires some refinement of the abstract model. The composition of the two components, $M_1||M_2$, is of course 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, rather than constructing the full composition.

6.3. Constructing the Product Subgraph

In this section we use the mc-graphs of $M_1\uparrow$ and $M_2\uparrow$ for constructing a subgraph for $M_1||M_2$, and by that avoid constructing 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. $\chi_f : N_1 \times N_2 \to B$ is a function defined by $\chi_f(n_1,n_2) = \chi_1(n_1) \oplus \chi_2(n_2)$.*
$E_f : (N_1 \times N_2) \times (N_1 \times N_2) \to B$ 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 an mc-graph and let $f : N \to B$ and $e : N \times N \to B$ 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)*** *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 an initial node $n_i^0 = (s_i^0 \vdash \varphi) \in N_i$. 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 (smallest) graph that obeys the following:*

- *$n_\|^0 = (s_1^0, s_2^0) \vdash \varphi$ 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. *$E_\|((n_1, n_2), (n_1', n_2')) = E_f((n_1, n_2), (n_1', n_2'))$.*

Recall that the $\oplus$ operation might be undefined on CPDBs (due to the properties of a partial bilattice, see Definition 3.6). The following lemma ensures that despite this property, $\chi_f(n)$, which is defined using $\oplus$, is defined for every $n \in N_\|$.

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

Proof: Assume $n = (s_1, s_2) \vdash \psi$. χ_1 is a correct mc-function of G_1, thus $\|\psi^*\|^{M_1\uparrow} = \chi_1(s_1 \vdash \psi)$. (Recall that ψ^* denotes the result of replacing every free occurrence of $Z \in Var$ in ψ by $fp(Z)$.) ψ^* is a closed L_μ formula, then since $M_1||M_2 \preceq M_1\uparrow$ we conclude that $\chi_1(s_1 \vdash \psi) \leq_\mathbf{i} \|\psi^*\|^{M_1||M_2}(s_1, s_2)$. Similarly, $\chi_2(s_2 \vdash \psi) \leq_\mathbf{i} \|\psi^*\|^{M_1||M_2}(s_1, s_2)$. As a result, $\chi_1(s_1 \vdash \psi) \oplus \chi_2(s_2 \vdash \psi) \leq_\mathbf{i} \|\psi^*\|^{M_1||M_2}(s_1, s_2)$, 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 truth order than $\bot$.

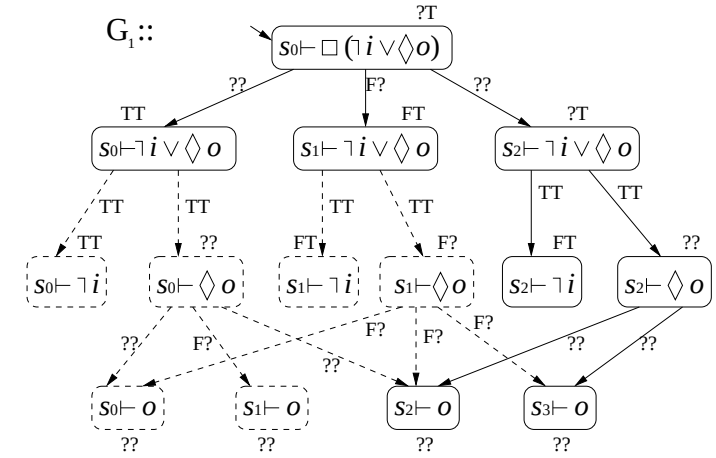
Product Graph Construction. The product graph is constructed by a top-down traversal on the mc-graphs of the two models, where, starting from the initial node, nodes that share the same formulas and whose states are weakly composable, will be considered. Whenever two non-terminal nodes n_1, n_2 are composed, 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 addition, 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 $G_\parallel$. Note that we compute χ_f and E_f only by need. In fact, when constructing $G_\parallel$ it suffices to consider the indefinite subgraphs $G_1^?$ and $G_2^?$ of G_1 and G_2 respectively (pruned to definite nodes). This is because 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, i, implies that there is a successor state where the output signal, o, holds. The mc-graphs of the lifted models $M_1\uparrow$ and $M_2\uparrow$ are described in Figure 7(a) on page 42 and Figure 7(b) on page 42 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 8 on page 43. 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. During the construction of the product graph, these nodes are temporarily created, but then removed on-the-fly, since they are covered. Therefore, they do not remain in the resulting product graph. The actual nodes that compose 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.

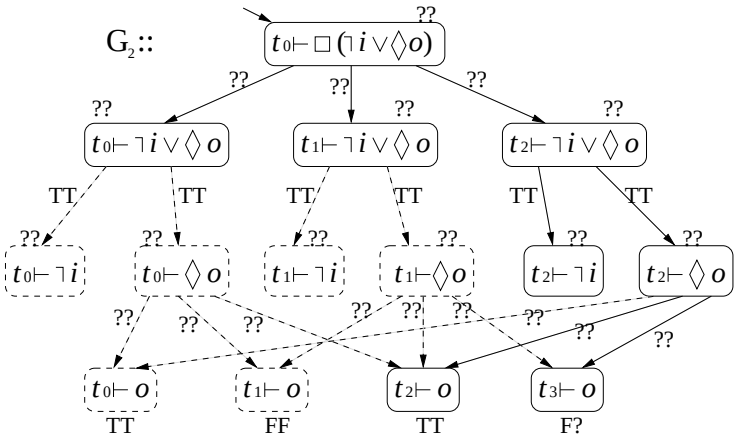
We accompany $G_\parallel$ with an initial mc-function, χ_I, for its terminal nodes, based on the mc-functions of the two mc-graphs. We use the following observation:

Observation 6.16. *Let $n = (s_1, s_2) \vdash \psi$ be a terminal node in $G_\parallel$. 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 6.17. *The initial mc-function χ_I of $G_\parallel$ is defined as follows. Let $n = (s_1, s_2) \vdash \psi \in N_\parallel$ be a terminal node. If it fulfills case (a) or (b) of Observation 6.16, then $\chi_I(n) = \chi_1(s_1 \vdash \psi) \oplus \chi_2(s_2 \vdash \psi)$. If it fulfills case (c), then*



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



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

Figure 7: mc-graphs of $M_1\uparrow$ and $M_2\uparrow$. Solid lines mark composed subgraph

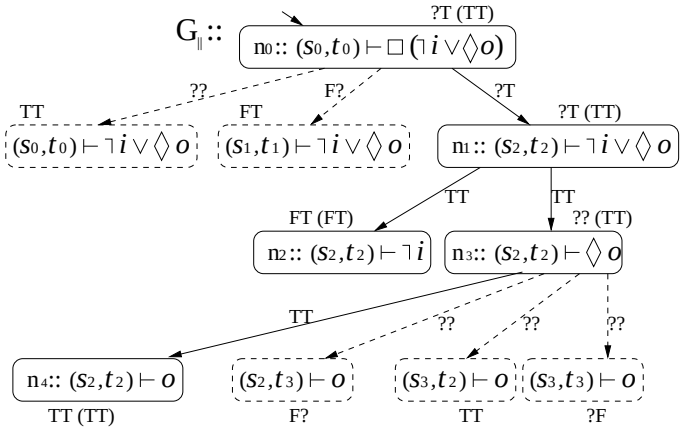


Figure 8: The product graph. Dashed nodes are covered. Solid lines mark actual product graph.

$\chi_I(n) = true$ 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 6.18. *Let G be the mc-graph over $M_1||M_2$, and let χ be its mc-function. The 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 6.18 we need the following Lemmas.

Lemma 6.19. *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). Then for every $(s_1, s_2) \in H$, $\chi_2(s_2 \vdash \psi) \leq_{\mathbf{i}} \chi_1(s_1 \vdash \psi)$.*

Proof: χ_1 is a semantically correct mc-function for G_1, thus $\|\psi^*\|^{M_1}(s_1) = \chi_1(s_1 \vdash \psi)$. (Recall that ψ^* denotes the result of replacing every free occurrence of $Z \in Var$ in ψ by $fp(Z)$.) 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, $\|\psi^*\|^{M_2}(s_2) \leq_{\mathbf{i}} \|\psi^*\|^{M_1}(s_1)$. Thus, $\chi_2(s_2 \vdash \psi) \leq_{\mathbf{i}} \chi_1(s_1 \vdash \psi)$ □

Lemma 6.20. *The initial mc-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 6.17, the initial mc-function 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 $\chi_i(s_i \vdash \psi) \leq_{\mathbf{i}} \chi((s_1, s_2) \vdash \psi)$ (Lemma 6.19). This means that $\chi_1(s_1 \vdash \psi) \oplus \chi_2(s_2 \vdash \psi) \leq_{\mathbf{i}} \chi((s_1, s_2) \vdash \psi)$. 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 6.18.

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 construction 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, $\chi_1(n_1) \oplus \chi_2(n_2) \leq_{\mathbf{i}} \chi(n)$, thus $\chi_f(n) \leq_{\mathbf{i}} \chi(n)$. This relation holds for every node in N, thus $\chi_f(n') \leq_{\mathbf{i}} \chi(n')$ and $\chi_f(n'') \leq_{\mathbf{i}} \chi(n'')$. 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 proved that χ_I is defined on all terminal nodes of $G_{\|}$, and $\chi_I(n) = \chi(n)$ for every terminal node. By Lemma 6.20, 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 if 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 $\chi_i(n_i) \leq_{\mathbf{i}} \chi(n)$ (Lemma 6.19). Thus $\chi_1(n_1) \oplus \chi_2(n_2) \leq_{\mathbf{i}} \chi(n)$. 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 6.18 imply that applying the **mc-algorithm** on $G_\parallel$ with χ_I results in a correct mc-function χ with respect to $G_\parallel$. This is because Theorem 6.18 ensures that $G_\parallel$ and χ_I fulfill all the properties listed in Theorem 5.5 as sufficient conditions for correctness of the **mc-algorithm** over subgraphs. Thus, $\chi(n^0_\parallel)$ 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 $G_\parallel$. Note that the full graph for $M_1||M_2$ is not constructed.

Example 6.21. *Consider the product graph in Figure 8 on page 43. By Theorem 6.18, this is a closed subgraph of the mc-graph over $M_1||M_2$ with mc-function χ. By Definition 6.17, the initial mc-function χ_I of $G_\parallel$ is: $\chi_I(n_4) = TT$ and $\chi_I(n_2) = FT$. Note that these are the only terminal nodes in $G_\parallel$.*
*Since this is a closed subgraph, then by Theorem 5.5, the **mc-algorithm** can be applied on it. The mc-function of $G_\parallel$ is described in Figure 8 on page 43, 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 data 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 $G_\parallel$ of the subgraphs $G_1^?$ and $G_2^?$.
2. Apply multi-valued model checking on $G_\parallel$ (with the initial mc-function).

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

Step 3: Refine: Consider the failure data returned by model checking $G_\parallel$ (where $\chi_\parallel(n_\parallel^0)$ 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 $R_i(s_i, s_i') \leq_{\mathbf{t}} R_{\bar{i}}(s_{\bar{i}}, s_{\bar{i}}')$, 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 subgraph(s) in which the edge is indefinite.

Go to Step 1(2) with the refined indefinite subgraphs.

Theorem 6.22. *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 data finding algorithm, if the answer is not definite, then the algorithm returns failure data. Based on Lemma 4.9, 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. □

7. Handling Full Distributive Bilattices

In the previous sections we have presented the compositional framework for abstract and concrete models which were defined over CPDBs. In the following section 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 [54] and YASM [36]. In both, the multi-valued structure used is the Belnap structure (Figure 1(a) and (b) on page 6). We will present our compositional framework over full distributive bilattices in light of these examples.

In the rest of the section we first describe mixed simulation and refinement of multi-valued models over full distributive 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 models.

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

In order to discuss the application of our compositional framework to full bilattices, we first need to revisit some of the definitions and algorithms provided for multi-valued models over CPDBs. We start by the notion of a mixed simulation relation which we used both to relate a concrete model and its abstractions, and also to relate two abstract models where one refines of the other. We then discuss refinement, and specifically which model checking results require refinement in this setting.

Relation between Concrete and abstract Models. In Section 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 $\mathcal{B}(\mathcal{D})= (D \times D, \leq_{\mathbf{i}}, \leq_{\mathbf{t}}, \neg)$, where the values which are actually used in the components of M_c 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 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, this means that the abstract model M_A is *not* an abstraction of the concrete model M_c, as it cannot be stated that M_c contains equally or more information than M_A (w.r.t. the information order). Still, we refer to M_A as "abstract" to reflect the property that it does not define a concrete model.

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 in any particular example depends on the way the abstract model is created, and lies beyond the scope of this paper.

Our starting point is therefore a concrete model M_c over a De Morgan algebra $\mathcal{D}$, and an abstract model M_A over the full distributive bilattice $\mathcal{B}(\mathcal{D})$, with some example-specific relation between them that ensures that every definite value obtained on M_A carries over to M_c.

Relation between Abstract Models. While we do not relate a concrete model defined over a De Morgan algebra $\mathcal{D}$ and its abstract model defined over $\mathcal{B}(\mathcal{D})$ by a mixed simulation relation, when discussing two *abstract* models defined over a bilattice $\mathcal{B}(\mathcal{D})$, we will again use the mixed simulation relation to describe the connection between the two models. 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 $\mathcal{B}(\mathcal{D})$. 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 $\mathcal{B}(\mathcal{D})$.

Refinement. The next issue to address is refinement. Namely, when refinement of a model defined over a bilattice $\mathcal{B}(\mathcal{D})$ is needed, and how it should be performed.

As discussed in Section 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. How should these elements be treated? Do we wish to refine a value which is definite but not highest in the information order? 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. Similarly, consistent and definite values give 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 aiming to refine indefinite values (consistent or not), the refinement algorithm described in Section 4 can be used for distributive bilattices as well. This algorithm will not try to refine a definite value. Thus, only transitions or atomic propositions with indefinite values will be suggested as failure data.

Example 7.1. *Figure 9 on page 49 presents the bilattice of the* 4×4 *structure. This bilattice is created from the Belnap structure (Figure 1(a),(b) on page 6). 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 1(g),(h)). Consider the model* M *in Figure 10 on page 49. 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*

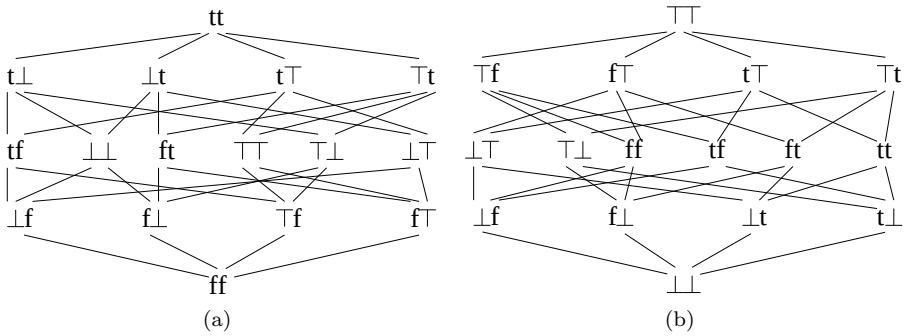


Figure 9: Truth (a) and information (b) orders of 4×4 structure

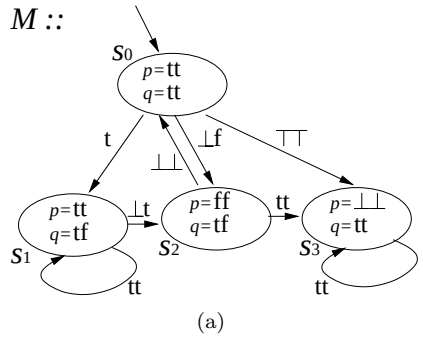


Figure 10: Model M over 4×4 structure

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 $\varphi = \Box p$, $\|\varphi\|^M = (\bot f \vee tt) \wedge (\bot t \vee f f) \wedge (\top\top \vee \bot\bot) = \bot t$. *This result is not definite. Based on the refinement algorithm presented in Section 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) [54]. 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 consist of primary inputs (which have no ingoing edges) and internal nodes, where internal nodes are divided to latches and combinational gates. A combinational gate node represents a Boolean operator. Given a directed edge (n_1, n_2) in a circuit C, we say that n_1 is an *input* of n_2. The circuit runs in time frames, defined by

clock cycles. Each node gets a Boolean value in each time frame. The value of a combinational gate node n is the result of applying its operator on its inputs. The value of a latch changes according to its input at each clock cycle. The graph of C may contain loops, but not combinational loops (i.e., loops that do not go through latches).

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 $v_i \in V$, $f_i : 2^V \times 2^{PI} \rightarrow \{0,1\}$. The transition functions summarize the value updates of latch nodes due to combinational paths of the circuit.

STE assertions are of the form $A \Longrightarrow G$, where A expresses constraints on circuit nodes at specific times, and G expresses requirements that should hold on circuit nodes at (possibly other) specific times. To verify STE assertions, the STE assertions are augmented into the circuit as described in [54].

In STE, a circuit node can get a value in the Belnap structure, $\mathcal{B}$= $(\{\top, \bot, t, f\}, \leq_{\mathbf{t}}, \leq_{\mathbf{i}}, \neg)$ (Figure 1(a),(b) on page 6). 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 $M = \langle \mathcal{B}, AP, S, s_0, R, \Theta \rangle$ where $\mathcal{B}$ 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, where $R(s, s') = t$ if $\forall v_i \in V, s'(v_i) = f_i(s\,|_V, s'\,|_{PI})$. 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 $a \in AP$, $\Theta(a)(s) = s(a)$.

We do not include here the construction of (abstract) STE models [54]. We do, however, discuss the characteristics of the Kripke models which correspond to the STE models constructed in order to verify STE assertions:

- 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 the 3-valued logic, thus 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.

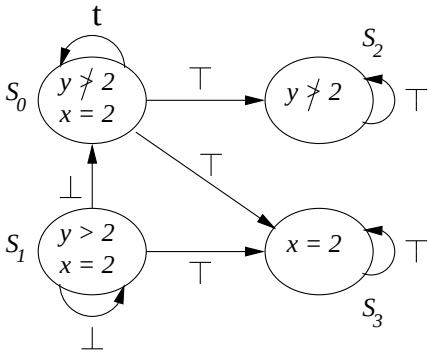


Figure 11: A 4-valued model

YASM - a Software Model-Checker [36]. The authors present a framework for 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 $\bot$ (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 YASM.

Example 7.2. *[36] Figure 11 on page 51 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. Based on the construction, 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 valuation of φ on the abstract model will not be $\top$. Moreover, if the valuation is t or f on the abstract model then the valuation 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 exists.

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

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 $\bot$ 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 (Definition 6.2) used in the definition of composition of models 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 Section 6, remains the same.

YASM. As described above, the atomic propositions are evaluated over t, f and $\bot$, 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.

8. Related Work

In this section we discuss some of the most closely related work in the areas of multi-valued model checking, abstraction-refinement and compositional verification.

Multi-Valued Model Checking. Multi-valued model checking [14, 12, 7] has many important applications within the verification framework. For example, 3-valued models are used to describe models with partial information and allow proving truth as well as falsity of formulas for the concrete models they represent [5, 56, 31, 32]. [2] uses a 6-valued logic for falsification of properties. 4-valued models can model disagreement and inconsistencies in software [36] and hardware (in STE) [54]. Generalization of 3-valued (and 4-valued) models are used to handle

inconsistent views of a system [24, 39]. 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 [22].

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, 43, 35, 7, 44]. As opposed to the reduction approach, the direct approach checks the property directly on the multi-valued structure [15, 13, 11, 57]. 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 [57].

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 for presenting a framework for creating precise abstractions. However, bilattices are not exploited there for refinement. In [2] 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 [3], with both information and truth orders, is used in software modelling ([36]). There, the 4 values of the logic are used to distinguish between unknowns resulting from abstraction (that represent lack of information) and unknowns resulting from the environment (that represent non-determinism). In STE the 4-valued Belnap logic is used for hardware modelling ([54]). 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.

Similarly to the works discussed, we also use bilattices to present any multi-valued logic. We define a mixed simulation relation between models based on bilattices. The mixed simulation for the 6-valued logic described in [2] 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, and a compositional framework.

The lifting of a De Morgan algebra $\mathcal{D}$ to a bilattice $\mathcal{B}(\mathcal{D})$ which contains both the concrete elements of $\mathcal{D}$ and their approximations (Definition 3.2) resembles the "intervals" abstraction of [48]. Their abstraction of an element is by means of a lower bound and an upper bound, where the "degree of truth" provided by an element of $\mathcal{B}(\mathcal{D})$ can be viewed as a lower bound on the approximated concrete element from $\mathcal{D}$, and the "degree of falsity" can be viewed as the negation of an upper bound.

Mixed Simulation Relations. 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 [45], 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 formulas of some logic (specifically, full μ-calculus formulas).

Mixed simulation for the 3-valued semantics is presented in [20, 29], which has the property of preserving the full μ-calculus. In [2] the authors present a relation similar to our mixed simulation for the 6-valued semantics. Their relation preserves temporal logic formulas consisting of both universal and existential quantifiers. The preservation of formulas is defined with respect to information order. However, they do not handle a general multi-valued framework, but rather a 6-valued one. Our work generalized the notion of mixed simulation for any multi-valued semantics. Similarly to [2], preservation of formulas is defined with respect to information order. In addition, they suggest refinement only if the result is the smallest element in the information order, $\bot$. In contrast, we allow refinement for any indefinite value in the bilattice.

Abstraction-Refinement. The traditional abstraction-refinement framework, referred to as CEGAR [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. [47, 51, 1] suggest abstraction-refinement mechanisms for the μ-calculus over 2-valued semantics, for *specific* abstractions. Model checking of abstract models for specifications in μ-calculus interpreted over the 3-valued semantics is investigated in [28, 30]. 3-valued game-based model checking for the full μ-calculus is suggested in [31, 32]. They also present an automatic refinement mechanism, which is based on finding a *failure cause*. [2] 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. 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 the components, the Assume-Guarantee (AG) paradigm [41, 53, 33] suggests how to verify one module based on assumptions about the behavior of its environment. The environment is then verified in order to guarantee that the assumptions are actually satisfied.

[19] proposed the *learning-based AG framework*. 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 (e.g., [8, 9, 27, 59, 25, 50, 34, 18, 52, 16]). These works are all designed for universal safety properties, with the exception of [25], which learns the full class of ω-regular languages.

[4] 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].

Other approaches for compositional reasoning are based on *interface automata* [21, 23] for the description of the dependencies between components. There, assumptions and guarantees on the different components of the system are given as automata. Composition of components is defined by relation between the assumptions and the guarantees. These works address refinement of components, which is adding details to a partially defined component, while preserving its interface. Though we also address an abstraction-refinement relation in a system, these works are very different from ours. These works look at manual refinement as part of the development process, whereas we are suggesting an automatic abstraction and refinement, and our goal is improving scalability of the verification tool.

[46] uses 3-valued model checking for modular verification. They consider feature-oriented modules, where the composition is via interfaces and has a more sequential nature. As a result, they only refer to unknown propositions and not to uncertainty in the transitions. A substantial part of their work is devoted to determining what information needs to be included in a feature's interface to support compositional reasoning.

In contrast to these approaches, the compositional technique presented in [58] is based on a 3-valued game for model checking. There, the model checking graph enables the sharing of auxiliary information between components.

We present a framework generalizing the compositional approach of [58] to handle multi-valued models. 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 [58].

9. Conclusion

In this paper 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-graphsand 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 algorithm for computing the mc-function and a specific cause finding algorithm (Algorithm 2.8 and Section 4). The computation of the mc-function and the cause finding algorithm can be replaced by any model checking algorithm.

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 [58] a compositional framework such as ours has been presented for the full L_μ 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 [2], along with a 6-valued structure (described in Figure 1(c),(d) on page 6). The structure is a CPDB, but since they use a logic with significantly different semantics, specific adaptations in some of the framework stages should be done.

Acknowledgement. The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no [321174].

References

[1] A. Asteroth, C. Baier, and U. Aßmann. Model checking with formula-dependent abstract models. In *CAV*, pages 155–168, 2001.

[2] T. Ball, O. Kupferman, and G. Yorsh. Abstraction for falsification. In *CAV'05*, pages 67–81, Edinburgh, Scotland, UK, July 2005.

[3] N. 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.

[4] M. G. Bobaru, C. S. Pasareanu, and D. Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In *Computer Aided Verification (CAV'08)*, volume 5123 of *LNCS*, Princeton, NJ, USA, July 2008. Springer.

[5] G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *CAV'99*, pages 274–287, 1999.

[6] G. Bruns and P. Godefroid. Temporal logic query checking. In *LICS'01*. IEEE, 2001.

[7] G. Bruns and P. Godefroid. Model checking with multi-valued logics. In *ICALP*, 2004.

[8] S. Chaki, E. M. Clarke, N. Sinha, and P. Thati. Automated assume-guarantee reasoning for simulation conformance. In *CAV'05*, pages 534–547, Edinburgh, Scotland, UK, July 2005.

[9] S. Chaki and O. Strichman. Optimized l*-based assume-guarantee reasoning. In *TACAS'07*, pages 276–291, Braga, Portugal, March 2007.

[10] W. Chan. Temporal-logic queries. In *CAV*, volume 1855 of *LNCS*, pages 450–463, 2000.

[11] M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. Multi-valued symbolic model-checking. *ACM Transactions on Software Engineering and Methodology*, 12:2003, 2003.

[12] M. Chechik, B. Devereux, S. Easterbrook, A. Y. C. Lai, and V. Petrovykh. Efficient multiple-valued model-checking using lattice representations. In *In Proc. 12th Int. Conf. on Concurrency Theory (CONCUR'01), volume 2154 of LNCS*, pages 451–465. Springer, 2001.

[13] M. Chechik, B. Devereux, A. Gurfinkel, and S. Easterbrook. Multi-valued symbolic model-checking. Technical Report CSRG-448, University of Toronto, April 2002.

[14] M. Chechik, S. Easterbrook, and V. Petrovykh. Model-checking over multi-valued logics. In *FME '01: Proceedings of the International Symposium of Formal Methods Europe on Formal Methods for Increasing Software Productivity*, pages 72–98, London, UK, 2001. Springer-Verlag.

[15] M. Chechik, A. Gurfinkel, and B. Devereux. chi-chek: A multi-valued model-checker. In *CAV*, pages 505–509, 2002.

[16] Y.-F. Chen, A. Farzan, E. M. Clarke, Y.-K. Tsay, and B.-Y. 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] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, pages 154–169, 2000.

[18] J. M. Cobleigh, G. S. Avrunin, and L. 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.

[19] J. M. Cobleigh, D. Giannakopoulou, and C. S. Pasareanu. Learning assumptions for compositional verification. In *TACAS'03*, volume 2619 of *LNCS*, pages 331–346, Warsaw, Poland, April 2003.

[20] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.

[21] L. de Alfaro and T. A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, Sept. 2001.

[22] L. de Alfaro, M. Z. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In S. Graf and M. Schwartzbach, editors, *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 395–410. Springer, 2000.

[23] L. Doyen, T. A. Henzinger, B. Jobstmann, and T. Petrov. Interface theories with component reuse. In *Proceedings of the 8th ACM International Conference on Embedded Software*, EMSOFT '08, pages 79–88, 2008.

[24] S. Easterbrook and M. Chechik. A framework for multi-valued reasoning over inconsistent viewpoints. In *ICSE'01*, pages 411–420, 2001.

[25] A. Farzan, Y.-F. Chen, E. M. Clarke, Y.-K. Tsay, and B.-Y. 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] M. Fitting. Bilattices are nice things, 2002.

[27] M. Gheorghiu, D. Giannakopoulou, and C. 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] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based model checking using modal transition systems. In *CONCUR*, pages 426–440, 2001.

[29] P. Godefroid and R. Jagadeesan. Automatic abstraction using generalized model checking. In *CAV'02*, pages 137–150, 2002.

[30] P. Godefroid and R. Jagadeesan. On the expressiveness of 3-valued models. In *VMCAI*, pages 206–222, 2003.

[31] O. Grumberg, M. Lange, M. Leucker, and S. Shoham. Don't know in the μ-calculus. In *VMCAI*, 2005.

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

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

[35] A. Gurfinkel and M. Chechik. Multi-valued model checking via classical model checking. In *CONCUR*, pages 263–277, 2003.

[36] A. Gurfinkel and M. Chechik. Why waste a perfectly good abstraction? In *TACAS'06*, pages 212–226, Vienna, Austria, April 2006.

[37] A. Gurfinkel, B. Devereux, and M. Chechik. Model exploration with temporal logic query checking. In *FSE'02*, pages 139–148. ACM, 2002.

[38] A. Gurfinkel, O. Wei, and M. Chechik. Systematic construction of abstractions for model-checking. In *VMCAI'06*, pages 381–397, 2006.

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

[40] H. Jain, D. Kroening, N. Sharygina, and E. M. Clarke. Word level predicate abstraction and refinement for verifying RTL Verilog. In *DAC'05*, pages 445–450, 2005.

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

[42] S. C. Kleene. *Introduction to metamathematics*. Bibl. Matematica. North-Holland, Amsterdam, 1952.

[43] B. Konikowska and W. Penczek. Reducing model checking from multi-valued CTL* to CTL*. In *CONCUR*, volume 2421 of *LNCS*, 2002.

[44] B. Konikowska and W. Penczek. Model checking multi-valued modal mu-calculus: Revisited. In *Proc. of CS&P'04*, 2004.

[45] O. Kupferman and Y. Lustig. Latticed simulation relations and games. In *ATVA'07*, pages 316–330, Tokyo, Japan, October 2007.

[46] H. C. Li, S. Krishnamurthi, and K. Fisler. Modular verification of open features using three-valued model checking. *Autom. Softw. Eng.*, 12(3):349–382, 2005.

[47] J. Lind-Nielsen and H. R. Andersen. Stepwise ctl model checking of state/event systems. In *CAV*, pages 316–327, 1999.

[48] D. Massé. Abstract domains for property checking driven analysis of temporal properties. In *Proceedings of the Tenth International Conference on Algebraic Methodology And Software Technology (AMAST'2004)*, volume 3116 of *Lecture Notes in Computer Sciences*, Stirling, UK, July 12–16 2004. Springer-Verlag, Berlin, Germany.

[49] Y. Meller, O. Grumberg, and S. Shoham. A framework for compositional verification of multi-valued systems via abstraction-refinement. In *Automated Technology for Verification and Analysis, 7th International Symposium (ATVA)*, pages 271–288, 2009.

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

[51] A. Pardo and G. D. Hachtel. Automatic abstraction techniques for propositional μ-calculus model checking. In *CAV*, pages 12–23, 1997.

[52] C. S. Pasareanu, D. Giannakopoulou, M. G. Bobaru, J. M. Cobleigh, and H. Barringer. Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. *Formal Methods in System Design*, 32(3), 2008.

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

[54] C.-J. H. Seger and R. E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. *Formal Methods in System Design*, 6(2), 1995.

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

[56] S. Shoham and O. Grumberg. A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. In *CAV'03*, volume 2725 of *LNCS*, pages 275–287, 2003.

[57] S. Shoham and O. Grumberg. Multi-valued model checking games. In *ATVA'05*, pages 354–369, 2005.

[58] S. Shoham and O. Grumberg. Compositional verification and 3-valued abstractions join forces. In *SAS'07*, volume 4634 of *LNCS*, Kongens Lyngby, Denmark, August 2007. Springer.

[59] N. Sinha and E. 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.

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