Compositional Verification and 3-Valued Abstractions Join Forces[☆]

Sharon Shoham, Orna Grumberg

Computer Science Department, Technion, Haifa, Israel

Abstract

Two of the most promising approaches to fighting the state explosion problem are abstraction and compositional verification. In this work we join their forces to obtain a novel fully automatic compositional technique that can determine the truth value of the full μ-calculus with respect to a given system.

Given a system $M = M_1 \| M_2$, we view each component M_i as an abstraction $M_i\uparrow$ of the global system. The abstract component $M_i\uparrow$ is defined using a 3-valued semantics so that whenever a μ-calculus formula φ has a definite value (true or false) on $M_i\uparrow$, the same value holds also for M. Thus, φ can be checked on either $M_1\uparrow$ or $M_2\uparrow$ (or both), and if any of them returns a definite result, then this result holds also for M. If both checks result in an indefinite value, the composition of the components needs to be considered. However, instead of constructing the composition of $M_1\uparrow$ and $M_2\uparrow$, our approach identifies and composes only the parts of the components in which their composition is necessary in order to conclude the truth value of φ. It ignores the parts which can be handled separately. The resulting model can potentially be significantly smaller than the full system.

We explain how our compositional approach can be combined with abstraction of the components, in order to further reduce the size of the checked components. The result is an incremental compositional abstraction-refinement framework, which resembles automatic Assume-Guarantee reasoning.

1. Introduction

Model checking [2] is a useful approach for verifying properties of systems. It is given a model M of a system and a temporal logic formula φ, describing a specification, and returns 'true' if the system satisfies the specification ($M \models \varphi$) and 'false', otherwise. The main disadvantage of model checking is the state explosion problem, which refers to its high space requirements. Two of the most promising approaches to fighting the state explosion problem are abstraction and compositional verification. In this work we join their forces to obtain a novel fully automatic compositional technique that can determine the truth value of the full μ-calculus with respect to a given system.

[☆] A preliminary version of this paper appeared in [1].

Email addresses: sharonsh@cs.tehnion.ac.il (Sharon Shoham), orna@cs.technion.ac.il (Orna Grumberg)

In compositional model checking one tries to verify parts of the system separately in order to avoid the construction of the entire system. To account for the dependencies between the components, the Assume-Guarantee (AG) paradigm [3, 4] suggests how to verify one module based on an *assumption* about the behavior of its environment, where the environment consists of the other system modules. The environment is then verified, in order to *guarantee* that it actually satisfies the assumption. Many of the works on compositional model checking are based on the AG paradigm and on learning [5–7] (see the related work section for more details). In contrast, our approach is based on techniques taken from the 3-valued game-based model checking for abstract models [8–10].

We first present our method for concrete systems, composed of concrete (unabstracted) components. We then extend it to abstract systems, in which one or both of the components have been abstracted (separately). In both cases we avoid the construction of the full system and compose only the parts of the components in which the value of the checked formula remained inconclusive. For simplicity we refer to systems that consist of two components $M_1 \| M_2$. However, our approach can be extended to the composition of n components. In our setting M_1 and M_2 are Kripke structures that synchronize on the joint labeling of the states. This means that a state of one model is composed with all the states of the other that agree with it on the joint labels. This composition is suitable for modeling synchronous systems with shared variables. In particular, it is suitable for hardware designs that synchronize on their inputs and outputs, since our models can be viewed as Moore machines [11]. The underlying ideas are applicable to other models as well, such as Labeled Transition Systems (LTSs), where components synchronize on their joint transitions and interleave their local transitions (see Section 6.1).

Given a system $M = M_1 \| M_2$, we view each component M_i as an abstraction $M_i\uparrow$ of the global system M, in which the values of the local (unshared) variables and the transitions of the other component are unknown. We consider the 3-valued semantics of the μ-calculus, in which the value of a formula in a model is either tt (true), ff (false), or $\perp$ (unknown). $M_i\uparrow$ is defined so that whenever a μ-calculus formula φ has a definite value (tt or ff) on $M_i\uparrow$, the same value holds also for M. Thus, φ can be checked on either $M_1\uparrow$ or $M_2\uparrow$ (or both), and if any of them returns a definite result, then this result holds also for M. Only if both checks result in $\perp$, the value of φ in M is unknown.

For the 3-valued abstraction, when the model checking returns $\perp$, the abstract model should be *refined* in order to eliminate the $\perp$ result. For our framework, a refinement could be achieved by composing $M_1\uparrow$ and $M_2\uparrow$. This, however, is not desired and not necessary. Instead, only the parts of the abstract models for which the model checking result is $\perp$ are identified and composed. The resulting refined model is often significantly smaller than the full system and is guaranteed to return the correct model checking result.

The advantage of our approach is that instead of constructing the composition of $M_1\uparrow$ and $M_2\uparrow$, it focuses on the parts of the components in which their composition is indeed necessary, and ignores the parts which can be handled separately. Furthermore, if a certain formula only depends on one component, then it can be resolved on this component alone while avoiding the composition altogether.

To further reduce the size of the checked components, we combine our compositional approach with abstraction. Abstraction not only reduces the state-space of the components, but also allows to handle infinite-state components by abstracting them into finite-state components. Given a system composed of two (or more) components, we first abstract each component separately. However, in order to guarantee preservation of both tt and ff we require that the common alphabet (e.g. common inputs and outputs for hardware designs) will not be abstracted. Only

local (unshared) variables can be abstracted. While this limits the amount of reduction that can be achieved by the abstraction on a single component, it enables additional reduction due to the compositional reasoning.

We propose an automatic construction of the initial abstraction for each component separately. We then proceed as before: we run a 3-valued model checking on each of the components. If both return $\perp$, then we identify and compose the parts where indefinite results were obtained, and apply 3-valued model checking to the composed model. While in the concrete case this step always terminates with a definite result, here we may obtain an indefinite result due to abstraction. In such a case, we follow [8–10] in finding the *cause* for the indefinite result on the composed model. However, the refinement itself is applied on each of the components separately. Moreover, we adopt the incremental approach of [8–10] and refine only the indefinite part of each component.

An abstraction of a component M_i (which comprises the environment of the other component) can be viewed as providing an assumption on M_i. From this point of view, when applying abstraction-refinement on one or both of the components, the result is an automatic mechanism for assumption generation, which is either symmetric (refers to both components) or asymmetric (abstracts only one component). In each iteration, more information about the component is revealed, by need – based on the cause for the indefinite result. This resembles iterative AG reasoning. The use of conservative abstractions guarantees that the assumption describes the component correctly (by construction). Thus unlike typical AG reasoning, this need not be verified.

Our approach is based on the 3-valued game for model checking of μ-calculus, suggested in [9, 10]. The game is played on a *game graph*, whose nodes are labeled by $s \vdash \psi$, where s is a state in the checked model and ψ is a subformula of the checked formula, s.t. the value of ψ in s is relevant for determining the model checking result. The model checking algorithm "colors" each node in the game graph by T, F, or ? iff the value of ψ in s is tt, ff or $\perp$, respectively. Recall that we first apply the model checking algorithm to each component separately. If the algorithm colors a node $s \vdash \psi$ of $M_1\uparrow$ with T (F), then it is guaranteed that every state in the composed system M, whose first component is s, satisfies (falsifies) ψ. A similar property holds for $M_2\uparrow$. Thus, when the model checking returns $\perp$ then only the subgraphs of nodes whose color is ? require further checking and are therefore composed. As such, the game-based approach provides a natural way of identifying and focusing on the places where the value of the checked formula remained inconclusive. Still, the underlying ideas are not restricted to the game-based approach. We also discuss the incorporation of similar ideas into a symbolic algorithm (see Section 6.2).

In summary, our contribution is threefold:

- We introduce a new ingredient to compositional model checking, which enhances its modularity. Namely, given a compositional system, our approach uses a 3-valued model checking game-graph as a means to identify and focus on the parts of the components in which their composition is indeed necessary to conclude the truth value of the checked property, due to dependencies between them. It uses the game-graph to exchange information between the components in these points, by need, and ignores the parts which can be handled separately. Thus, it avoids the construction of the full composition. Furthermore, if a certain formula only depends on one component, then it is resolved on this component alone while avoiding the composition altogether. Our technique is orthogonal to the AG approach, and can also be applied when the composed system consists of a component and

an assumption on its environment.

- We develop a compositional, fully automatic, abstraction-refinement framework, which has some resemblance to iterative AG-reasoning, but benefits from the modular model checking described above. The refinement is also applied to each component separately. In addition, the abstraction-refinement is incremental in the sense that results from previous iterations are re-used. From the AG point of view, our compositional abstraction-refinement can be viewed as a new, automatic, mechanism for assumption generation, which uses the power of abstraction-refinement.

- Finally, unlike most automatic AG approaches, which are limited to universal safety properties, our technique is applicable to the *full* μ-calculus.

Related Work

Recently, [5] followed by [6, 7], considered automatic assumption generation for AG reasoning. They use *learning* algorithms for finite automata in order to automatically produce suitable assumptions for an AG rule. A similar approach is taken in [12], where the AG rule used is symmetric. Assumption generation in a more general setting (not necessarily for AG reasoning) is addressed in [13]. The work of [14] on interface synthesis for application programs can also be seen as assumption generation. These works are all restricted to *universal safety* properties (either in a linear time or a branching time setting). More recently, [15] extended the learning-based approach to liveness properties as well (in a linear time setting), by proposing a learning algorithm for the full class of ω-regular languages. The learning algorithms used in these works also perform some kind of an abstraction-refinement. However, these algorithms are not specifically tailored for verification. They do not always maintain a conservative abstraction of the environment. As such, the assumption sometimes needs to be weakened and sometimes needs to be strengthened. In our case an assumption (abstraction) should never be weakened. Following our preliminary work [1], an AG approach based on abstraction-refinement was suggested in [16]. Similarly to our approach, they always maintain a conservative abstraction of the environment. However, similarly to most of the learning-based approaches, their work is limited to linear-time safety properties. Most importantly, our approach is applicable to the *full* μ-calculus. Moreover, we increase the modularity of the model checking step by using the game-based approach, which also enables an incremental analysis.

The game-based model checking enables us to identify the places where the value of a subformula in a component's state is the same for *all* environments. We exploit this information to reduce the model checking instance of the entire system. Other authors have also used similar information for reductions. In [17] the authors merge component's states that share the same value for a given CTL formula in all environments, thus minimizing the component. In [18] the authors use reachability and controllability information about the concrete components (gathered via game-theoretic techniques) in order to construct abstract components for *invariance* properties. The composition of the abstract components is then computed and model checked. We, on the other hand, do not try to minimize each component. Instead, the game-graph enables us to prune parts of each component's model checking instance whose effect was already taken into consideration. As a result, we reduce the state space exploration of the entire system. This is applicable even if no states of the individual components can be merged.

[19] uses controllability information to speed up falsification of invariance properties. They identify unpreventable violations of the property based on each component separately, which enables to prune the state space exploration of the compound system before a violation is actually

encountered. The authors state that their method can be extended to arbitrary LTL properties. However, they only use controllability information w.r.t. the entire formula. Our approach enables to gather information about subformulas as well, and thus can result in more substantial reductions. In addition, our approach is aimed at both verification and falsification (with a 3-valued semantics) and is applicable to a full branching time logic.

[20] also 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 our case, we use the game graph for sharing such auxiliary information about the individual components.

In [21] the authors suggest to use game structures to reason about composition of components. [22, 23] suggest abstraction-refinement frameworks for such models, w.r.t. alternating time temporal logics, which enable to describe properties of the interaction between components. We are interested in properties of the compound system, thus the focus in these works is different. In addition, they abstract each component separately and then model check the entire system. The model checking step is not modular.

[24] develops a compositional counterexample-guided abstraction refinement for a universal temporal logic (which extends ACTL). In their approach, the abstraction and the refinement steps are performed on each component separately, but the model checking step is done on the entire (abstract) system. In our approach, the model checking step is also compositional, and the properties considered are not limited to a universal logic.

Organization of the paper

The rest of the paper is organized as follows. In the next section we give the necessary background for the μ-calculus, 3-valued abstraction and 3-valued game-based model checking. In Section 3, we discuss the properties of the game-based model checking which set the basis for our compositional algorithm. Our compositional model checking is then presented in Section 4 for concrete components, and extended to a compositional abstraction-refinement algorithm in Section 5. Section 6 describes extensions of the algorithm to the use of Labeled Transition Systems and to a symbolic algorithm. Finally, we discuss some conclusions in Section 7.

2. Preliminaries

μ-calculus. [25] Let AP be a finite set of atomic propositions and $\mathcal{V}$ a set of propositional variables. The set of literals over AP is $Lit = AP \cup \{\neg p : p \in AP\}$. We identify $\neg\neg p$ with p. The logic *μ-calculus* in *negation normal form* over AP is defined by:

$$\varphi \ ::= \ l \mid \Box\varphi \mid \Diamond\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid Z \mid \mu Z.\varphi \mid \nu Z.\varphi$$

where $l \in Lit$ and $Z \in \mathcal{V}$. Intuitively, $\Box$ stands for "all successors", and $\Diamond$ stands for "exists a successor". μ denotes a least fixpoint, whereas ν denotes greatest fixpoint. We will also write η for either μ or ν. Let $\mathcal{L}_\mu$ denote the set of *closed* formulas generated by the above grammar, where the fixpoint quantifiers μ and ν are variable binders. We assume that formulas are well-named, i.e. no variable is bound more than once in any formula. Thus, every variable Z *identifies* a unique subformula $fp_\varphi(Z) = \eta Z.\psi$ of φ, where the set $Sub(\varphi)$ of *subformulas* of φ is defined in the usual way.

Concrete Semantics. Concrete systems are typically modelled as *Kripke structures*. A Kripke structure [2] is a tuple $M = (AP, S, s^0, R, L)$, where AP is a finite set of atomic propositions, S is a finite set of states, $s^0 \in S$ is an initial state, $R \subseteq S \times S$ is a transition relation, and $L : S \to 2^{Lit}$ is a labeling function, such that for every state s and every $p \in AP$, exactly one of p and $\neg p$ is in $L(s)$.

The *concrete semantics* $[\![\varphi]\!]^M$ of a closed formula $\varphi \in \mathcal{L}_\mu$ over AP w.r.t. a Kripke structure $M = (AP, S, s^0, R, L)$ is a mapping from S to $\{\mathrm{tt}, \mathrm{ff}\}$.

To handle subformulas which are not closed, an *environment* $\rho : \mathcal{V} \to (S \to \{\mathrm{tt}, \mathrm{ff}\})$, which explains the meaning of free variables, is introduced. $[\![\varphi]\!]^M_\rho$ is defined inductively, for every μ-calculus formula. We order the truth values by $\mathrm{ff} \leq \mathrm{tt}$. Then the functions in $S \to \{\mathrm{tt}, \mathrm{ff}\}$ form a complete lattice under pointwise ordering, i.e., for $g, g' : S \to \{\mathrm{tt}, \mathrm{ff}\}$, $g \sqsubseteq g'$ iff $\forall s \in S : g(s) \leq g'(s)$. Joins and meets in this lattice are denoted $g \sqcup g'$ and $g \sqcap g'$ resp. We denote with $\rho[Z \mapsto g]$ the environment that maps Z to g and agrees with ρ on all other arguments. In the following definition $f = \lambda g.[\![\varphi]\!]^M_{\rho[Z \mapsto g]}$ is an element of $(S \to \{\mathrm{tt}, \mathrm{ff}\}) \to (S \to \{\mathrm{tt}, \mathrm{ff}\})$ and $\mathrm{gfp}(f)$, $\mathrm{lfp}(f)$ stand for the greatest and least fixpoints of f. These fixpoints exist according to [26], since the functional f is monotone w.r.t. the order $\sqsubseteq$.

$$
\begin{array}{lcl}
[\![l]\!]^M_\rho & := & \lambda s. \begin{cases} \mathrm{tt}, & \text{if } l \in L(s) \\ \mathrm{ff}, & \text{if } \neg l \in L(s) \end{cases} \\
[\![\Box\varphi]\!]^M_\rho & := & \lambda s. \begin{cases} \mathrm{tt}, & \text{if } \forall t \in S, \text{ if } sRt \text{ then } [\![\varphi]\!]^M_\rho(t) = \mathrm{tt} \\ \mathrm{ff}, & \text{if } \exists t \in S \text{ s.t. } sRt \text{ and } [\![\varphi]\!]^M_\rho(t) = \mathrm{ff} \end{cases} \\
[\![\Diamond\varphi]\!]^M_\rho & := & \lambda s. \begin{cases} \mathrm{tt}, & \text{if } \exists t \in S, \text{ s.t. } sRt \text{ and } [\![\varphi]\!]^M_\rho(t) = \mathrm{tt} \\ \mathrm{ff}, & \text{if } \forall t \in S \text{ if } sRt \text{ then } [\![\varphi]\!]^M_\rho(t) = \mathrm{ff} \end{cases} \\
[\![\varphi_1 \wedge \varphi_2]\!]^M_\rho & := & [\![\varphi_1]\!]^M_\rho \sqcap [\![\varphi_2]\!]^M_\rho \\
[\![\varphi_1 \vee \varphi_2]\!]^M_\rho & := & [\![\varphi_1]\!]^M_\rho \sqcup [\![\varphi_2]\!]^M_\rho \\
[\![Z]\!]^M_\rho & := & \rho(Z) \\
[\![\mu Z.\varphi]\!]^M_\rho & := & \mathrm{lfp}(\lambda g.[\![\varphi]\!]^M_{\rho[Z \mapsto g]}) \\
[\![\nu Z.\varphi]\!]^M_\rho & := & \mathrm{gfp}(\lambda g.[\![\varphi]\!]^M_{\rho[Z \mapsto g]})
\end{array}
$$

Note that for a closed formula φ, $[\![\varphi]\!]^M_\rho = [\![\varphi]\!]^M_{\rho'}$, for any environments ρ, ρ'. Thus, when closed formulas are considered, we drop the environment from the semantic brackets, and simply refer to $[\![\varphi]\!]^M$.

$[\![\varphi]\!]^M(s) = \mathrm{tt}$ $(= \mathrm{ff})$ means that the formula φ is true (false) in the state s of the Kripke structure M. If $[\![\varphi]\!]^M(s^0) = \mathrm{tt}$ $(= \mathrm{ff})$, we say that M satisfies (falsifies) φ, denoted $M \models \varphi$ $(M \not\models \varphi)$.

3-Valued Abstraction. In the context of abstraction, *Kripke Modal Transition Systems* [27, 28], which extend *Modal Transition Systems* [29], are often used as abstract models that preserve the μ-calculus.

Definition 2.1. *A* Kripke Modal Transition System *(***KMTS***) is a tuple $M = (AP, S, s^0, R^+, R^-, L)$, where AP, S and s^0 are defined as before, $R^+, R^- \subseteq S \times S$ are must and may transition relations (resp.) such that $R^+ \subseteq R^-$, and $L : S \to 2^{Lit}$ is a labeling function such that for every state s and $p \in AP$,* at most *one of p and $\neg p$ is in $L(s)$.*

The *3-valued semantics* $[\![\varphi]\!]^M_3$ of a closed formula $\varphi \in \mathcal{L}_\mu$ w.r.t. a KMTS M is a mapping from S to $\{\mathrm{tt}, \mathrm{ff}, \bot\}$ [27, 30]. As in the concrete case, an environment $\rho : \mathcal{V} \to (S \to \{\mathrm{tt}, \mathrm{ff}, \bot\})$

is introduced to handle subformulas that are not closed. $[\![\varphi]\!]^M_{\rho 3}$ is then defined inductively. The concrete semantics of the logical operators $\wedge, \vee$ and of the fixpoints extends to the 3-valued case by extending the ordering of the truth values to ff $\leq \perp \leq$ tt and extending the order $\sqsubseteq$ of the functions in $S \to \{\text{tt}, \text{ff}, \perp\}$ accordingly. The semantics of the literals and the modalities is extended to the 3-valued case as follows.

$$[\![l]\!]^M_{\rho 3} := \lambda s. \begin{cases} \text{tt}, & \text{if } l \in L(s) \\ \text{ff}, & \text{if } \neg l \in L(s) \\ \perp, & \text{otherwise} \end{cases}$$

$$[\![\Box\psi]\!]^M_{\rho 3} := \lambda s. \begin{cases} \text{tt}, & \text{if } \forall t \in S, \text{ if } sR^-t \text{ then } [\![\psi]\!]^M_{\rho 3}(t) = \text{tt} \\ \text{ff}, & \text{if } \exists t \in S \text{ s.t. } sR^+t \text{ and } [\![\psi]\!]^M_{\rho 3}(t) = \text{ff} \\ \perp, & \text{otherwise} \end{cases}$$

and dually for $\Diamond\psi$ when exchanging tt and ff. When closed formulas are considered, we omit ρ from the notation. The notations $M \models \varphi$ and $M \not\models \varphi$ are used for KMTSs as well. In addition, if $[\![\varphi]\!]^M_3(s^0) = \perp$, the value of φ in M is indefinite.

The following definition formalizes the relation between two KMTSs that guarantees preservation of μ-calculus formulas w.r.t. the 3-valued semantics.

Definition 2.2 (Mixed Simulation). *[28, 31] Let $M_1 = (AP, S_1, s^0_1, R^+_1, R^-_1, L_1)$ and $M_2 = (AP, S_2, s^0_2, R^+_2, R^-_2, L_2)$ be two KMTSs, both defined over AP. We say that $H \subseteq S_1 \times S_2$ is a* mixed simulation *from M_1 to M_2 if $(s_1, s_2) \in H$ implies the following:*

1. $L_2(s_2) \subseteq L_1(s_1)$.
2. *if $s_1 R^-_1 s'_1$, then there is some $s'_2 \in S_2$ such that $s_2 R^-_2 s'_2$ and $(s'_1, s'_2) \in H$.*
3. *if $s_2 R^+_2 s'_2$, then there is some $s'_1 \in S_1$ such that $s_1 R^+_1 s'_1$ and $(s'_1, s'_2) \in H$.*

If there is a mixed simulation H s.t. $(s^0_1, s^0_2) \in H$, then M_2 abstracts *M_1, denoted $M_1 \preceq M_2$.*

In particular, Def. 2.2 can be applied to a (concrete) Kripke structure M_C and an (abstract) KMTS M_A, by viewing the Kripke structure as a KMTS where $R^+ = R^- = R$. For a KMTS M_A and a concrete model M_C such that $M_C \preceq M_A$ we also say that M_A is an *abstract model* of M_C, or that M_C is *represented* by M_A. Moreover, if $(s_c, s_a) \in H$, then we say that the abstract state s_a *represents* the concrete state s_c. For a Kripke structure, the 3-valued semantics agrees with the concrete semantics. Thus, preservation of $\mathcal{L}_\mu$ formulas is guaranteed by the following theorem.

Theorem 2.3. *[28] Let $H \subseteq S_1 \times S_2$ be the mixed simulation relation from a KMTS M_1 to a KMTS M_2. Then for every $(s_1, s_2) \in H$ and every $\varphi \in \mathcal{L}_\mu$ we have that $[\![\varphi]\!]^{M_2}_3(s_2) \neq \perp \Rightarrow [\![\varphi]\!]^{M_1}_3(s_1) = [\![\varphi]\!]^{M_2}_3(s_2)$.*

Thus, the 3-valued semantics preserves both satisfaction (tt) and refutation (ff) from an abstract KMTS to a concrete model represented by it. $\perp$, on the other hand, means that the truth value over the concrete model is unknown and can be either tt or ff.

2.1. Abstract Model Checking

A 3-valued game-based model checking for the μ-calculus over KMTSs was defined in [9, 10]. They introduce 3-valued parity games and translate the 3-valued model checking problem into the problem of determining the winner in a 3-valued model checking game, which is a

special case of a 3-valued parity game. Model checking is then performed by solving the game via a coloring algorithm.

The games are defined below. For our purposes, it is mainly important to understand the definition of the game graph, as well as the correctness of its coloring, formulated in Thm. 2.7. The rest of the details are mainly needed in order to understand the properties of the coloring investigated in Section 3.

3-Valued Parity Games. A *3-valued parity game* [9, 10] $\Gamma = (G, \Theta)$ has a *game graph* $G = (n^0, N_0, N_1, N_{tie}, E^+, E^-)$ s.t. N_0, N_1 and N_{tie} are disjoint sets of nodes. Let $N := N_0 \cup N_1 \cup N_{tie}$. Then $E^+ \subseteq E^- \subseteq (N \setminus N_{tie}) \times N$ are sets of must and may edges, meaning that every $n \in N_{tie}$ is a *terminal node*, i.e. has no outgoing edges. $n^0 \in N$ is the *initial node*. $\Theta : N \to \mathbb{N}$ is a *priority function* that maps each node $n \in N$ to a *priority*.

The 3-valued parity game is played by two players: Player 0 and Player 1. A *play* in the game starting at node n is a maximal sequence of nodes $n_0, n_1, \ldots$, where $n_0 = n$ and if $n_i \in N_\sigma$ (for $\sigma \in \{0, 1\}$), then Player σ moves from n_i to n_{i+1} using an edge $(n_i, n_{i+1}) \in E^-$. Thus, the edges of the game graph denote the possible *moves* in the game. The play is called σ*-consistent* iff Player σ chooses only moves (edges) that are (also) in E^+. A σ-consistent play is *winning* for Player σ if

- it is finite and ends in $N_{1-\sigma}$ or
- it is infinite and the maximal priority occurring infinitely often is even when $\sigma = 0$ or odd when $\sigma = 1$.

All other plays are a *tie*. Note that terminal nodes in N_σ are losing for Player σ.

A (memoryless) strategy for Player σ in the 3-valued parity game $\Gamma = (G, \Theta)$ is a function $\zeta : N_\sigma \to N$ such that for all $n \in N_\sigma$ and $n' \in N$: $\zeta(n) = n'$ implies that Player σ can move from n to n', i.e. $(n, n') \in E^-$. A play $n_0, n_1 \ldots$ is said to *conform* to ζ if for all $k \in \mathbb{N}$, s.t. $n_k \in N_\sigma$: $n_{k+1} = \zeta(n_k)$. A strategy ζ for Player σ is a *winning strategy* starting at node n if every play that starts from n and conforms to ζ is won by Player σ. If such a strategy exists then Player σ is the *winner* in the game starting at n. A winning strategy, resp. the winner, *in the game* is a winning strategy, resp. the winner, starting at the initial node of the game graph.

3-Valued Model Checking Game. Let $M = (AP, S, s^0, R^+, R^-, L)$ be a KMTS and $\varphi \in \mathcal{L}_\mu$. The *3-valued model checking game* $\Gamma_{M\times\varphi} = (G_{M\times\varphi}, \Theta_{M\times\varphi})$ for M and φ is a 3-valued parity game, where Player 0 takes the role of the *verifier* and Player 1 takes the role of the *falsifier*. The game is designed such that Player 0 is the winner iff the model checking result is tt, Player 1 wins iff the result is ff. Otherwise neither of the players wins and the model checking result is $\perp$.

Game Graph. The game graph $G_{M\times\varphi}$, or in short G, of the 3-valued model checking game presents all the information "relevant" for the model checking. The set of nodes N is a subset of $S \times Sub(\varphi)$, with $n^0 = s^0 \vdash \varphi \in N$. The (rest of the) nodes and the edges are defined by the rules of Fig. 1, with the meaning that whenever $n \in N$ is of the form of the upper part of the rule, the result in the lower part of the rule is also a node $n' \in N$ and $E^-(n, n')$. Moreover, $E^+(n, n')$ holds as well in all cases except for an application of the rules in the second column with a model's transition $(s, t) \in R^- \setminus R^+$.

The nodes of G are classified as $\wedge$, $\vee$, $\Box$, $\Diamond$, or literal nodes, based on their subformulas. Nodes whose subformulas are of the form Z or $\eta Z.\psi$ are *deterministic* – they have exactly one outgoing edge.

$$\frac{s \vdash \psi_0 \vee \psi_1}{s \vdash \psi_i}\ 0 : i \in \{0,1\} \qquad \frac{s \vdash \Diamond\psi}{t \vdash \psi}\ 0 :\ sR^+t \text{ or } sR^-t \qquad \frac{s \vdash \eta Z.\psi}{s \vdash Z}\ 0$$

$$\frac{s \vdash \psi_0 \wedge \psi_1}{s \vdash \psi_i}\ 1 : i \in \{0,1\} \qquad \frac{s \vdash \Box\psi}{t \vdash \psi}\ 1 :\ sR^+t \text{ or } sR^-t \qquad \frac{s \vdash Z}{s \vdash \psi}\ 0 : \text{if } fp_\varphi(Z) = \eta Z.\psi$$

Figure 1: Rules for the construction of the game graph $G_{M\times\varphi}$

Each rule in Fig. 1 is marked by 0 / 1 to indicate which player makes the corresponding move. This determines if the source node belongs to N_0 or N_1. Intuitively, the move (outgoing edge) that the player chooses from the node $s \vdash \psi \in N$ presents a "subgoal" that the player defines for verifying or falsifying ψ in s. The rules that correspond to $\vee$, $\wedge$, $\Diamond$ and $\Box$ nodes (shown in the first and second columns of Fig. 1) present a choice which the player can make. For example, in a $\vee$-node $s \vdash \psi_0 \vee \psi_1$ Player 0, the verifier, chooses the subformula that she intends to verify in s. In a $\Diamond$-node $s \vdash \Diamond\psi$ she chooses a successor of s in which she intends to verify ψ. In $\wedge$ and $\Box$ nodes Player 1, the falsifier makes similar choices for falsification. Since no choice is possible in the deterministic nodes (third column), we arbitrarily assign them to one player, let us say 0. Thus, $\vee$, $\Diamond$ and deterministic nodes belong to N_0, whereas $\wedge$ and $\Box$ nodes belong to N_1.

The literal nodes, of the form $s \vdash l$, are divided between N_0, N_1 and N_{tie} based on the truth value of the literal l in the state s: if the truth value is tt then the node belongs to N_1 (i.e., Player 0 wins in it); if the truth value is ff then the node belongs to N_0 (i.e., Player 1 wins in it); otherwise the node belongs to N_{tie}.

Priority Function. The priority function $\Theta_{M\times\varphi}$, or in short Θ, of the 3-valued model checking game is defined as follows. Given variables X, Y we write $X \prec_\varphi Y$ if Y occurs freely in $fp_\varphi(X)$, and $X <_\varphi Y$ if (X, Y) is in the transitive closure of $\prec_\varphi$. Let $Z_1, \ldots, Z_n$ be all the variables occurring in φ. They are partially ordered by the relation $\leq_\varphi$. Note that it is possible to assign to each variable a number $\Theta(Z_i)$ s.t. for all $i, j = 1, \ldots, n$:

- $\Theta(Z_i)$ is even iff Z_i is of type ν;
- $\Theta(Z_i) \leq \Theta(Z_j)$ whenever $Z_i \leq_\varphi Z_j$.

The priorities on the nodes are assigned as follows:

$$\Theta(s \vdash \psi) \ := \ \begin{cases} \Theta(Z) & \text{if } \psi = Z \\ 0 & \text{otherwise} \end{cases}$$

The following theorem formalizes the correctness of the 3-valued model checking game. For a (possibly not closed) subformula ψ of φ, ψ^* denotes[1] the result of replacing every free occurrence of $Z \in \mathcal{V}$ in ψ by $fp_\varphi(Z)$. Note that if ψ is closed, then $\psi^* = \psi$.

Theorem 2.4. *[9, 10] Let $\Gamma_{M\times\varphi}$ be the 3-valued model checking game for a KMTS M and $\varphi \in \mathcal{L}_\mu$. Then for every $n = s \vdash \psi \in N$:*

[1] In fact, ψ^* is parameterized by φ. However, to simplify the notation we omit φ. We only use this notation when φ is clear from the context.

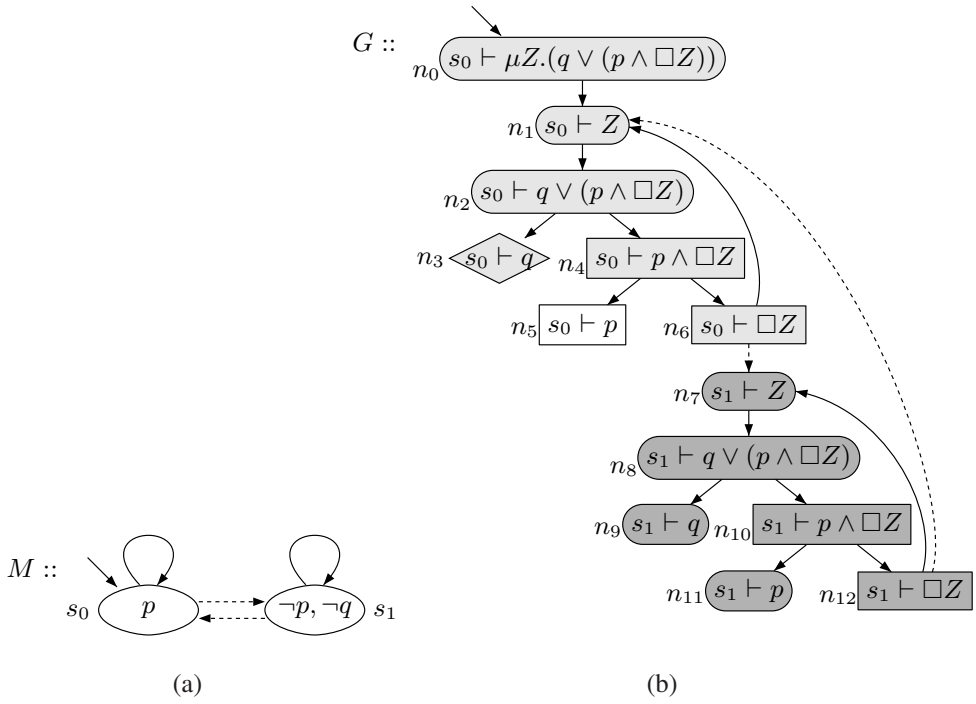


Figure 2: (a) A KMTS, and (b) a game graph. Dashed transitions in M denote may transitions which are not must transitions. Similarly for the edges in G. Round nodes in G denote nodes in N_0, whereas rectangular nodes are in N_1. Node n_3, shaped as a diamond, is in N_{tie}. The colors of the nodes in G reflect the coloring function of G: white stands for T, dark gray stands for F and light gray stands for ?.

1. $[\![\psi^*]\!]_3^M(s) = tt$ *iff Player* 0 *has a winning strategy starting at n.*
2. $[\![\psi^*]\!]_3^M(s) = ff$ *iff Player* 1 *has a winning strategy starting at n.*
3. $[\![\psi^*]\!]_3^M(s) = \bot$ *iff neither of the players has a winning strategy starting at n.*

In particular, if the winner in the 3-valued model checking game is Player 0 then the model checking result is tt, if it is Player 1 then the result is ff and if neither of the players has a winning strategy, then the result is $\bot$.

Example 2.5. Fig. 2(b) presents an example of the game graph G of the 3-valued model checking game for the KMTS M from Fig. 2(a) and the formula $\varphi = \mu Z.(q \vee (p \wedge \Box Z))$, which is equivalent to the CTL formula $A(pUq)$, meaning "in all paths, p holds until q holds". The priority function assigns all nodes in G priority 0, except for $n_1 = s_0 \vdash Z$ and $n_7 = s_1 \vdash Z$ which are assigned priority 1, since the fixpoint formula of Z in φ is of type μ, making its priority odd. The coloring of the nodes will be explained in Example 2.8.

Coloring Algorithm. The model checking algorithms of [9, 10] can be viewed as coloring algorithms[2] that label (color) each node $n = s \vdash \psi$ in the game graph of the 3-valued model checking game by T, F, ? depending on the player that has a winning strategy in the game,

[2]We refer to the model checking algorithms of [9, 10] as coloring algorithms although they were not originally described in these terms.

or equivalently depending on the truth value of ψ in the state s in M (based on the 3-valued semantics). The result of the coloring is a *3-valued coloring function* $\chi : N \to \{T, F, ?\}$.

In both [9] and [10] the coloring is performed by solving the 3-valued parity game for model checking $\Gamma_{M\times\varphi}$, where each color stands for a possible result (winner) in the game. The algorithm of [9] is a generalization of Zielonka's algorithm for solving (2-valued) parity games. In [10], the 3-valued parity game is reduced into two (2-valued) parity games, improving the coloring's complexity. In fact, any other algorithm for solving 3-valued parity games can be used as well.

For simplicity, we refer to these algorithms as coloring algorithms which are applied on the game graph $G_{M\times\varphi}$ alone. This is justified by the observation that in the context of the 3-valued model checking game, the game graph carries all the information regarding the game since the priorities of the nodes are determined by their subformulas. We therefore do not explicitly mention the priority function.

The following formalizes the correctness of the coloring w.r.t. the 3-valued model checking problem.

Definition 2.6. *Let $G_{M\times\varphi}$ be the game graph of the 3-valued model checking game for a KMTS M and $\varphi \in \mathcal{L}_\mu$. A (possibly partial) coloring function $\chi : N \to \{T, F, ?\}$ for $G_{M\times\varphi}$ (or its subgraph) is* correct *if for every $s \vdash \psi \in N$, whenever $\chi(s \vdash \psi)$ is defined, then:*

1. $[\![\psi^*]\!]_3^M(s) = tt$ *iff* $\chi(s \vdash \psi) = T$.
2. $[\![\psi^*]\!]_3^M(s) = ff$ *iff* $\chi(s \vdash \psi) = F$.
3. $[\![\psi^*]\!]_3^M(s) = \bot$ *iff* $\chi(s \vdash \psi) = ?$.

Theorem 2.7. *[9, 10] Let χ_F be the (total) coloring function returned by the coloring algorithm of [9] or [10] for $G_{M\times\varphi}$. Then χ_F is correct.*

Moreover, in both cases, the coloring of the nodes reflects the 3-valued semantics of the logic: A $\wedge$-node or a $\Box$-node is colored T iff all its may sons are colored T (and in particular if it has no may sons), it is colored F iff it has a must son which is colored F, and otherwise it is colored ?. Dually for a $\vee$-node or a $\Diamond$-node when exchanging T and F. The color of $s \vdash l$ for $l \in Lit$ is T iff $l \in L(s)$, F iff $\neg l \in L(s)$, and ? otherwise.

Example 2.8. Fig. 2(b) also demonstrates the result of the coloring and its correspondence with the semantics. For example, the node $n_3 = s_0 \vdash q$ in G is colored ?, which reflects the fact that neither q, nor $\neg q$ label the state s_0 in M, which makes the value of q in s_0 indefinite. Similarly, the initial node $n_0 = s_0 \vdash \varphi$ in G is colored ?. Indeed, the value of $\varphi = \mu Z.(q \vee (p \wedge \Box Z))$ in the state s_0 of the KMTS M from Fig. 2(a) is $\bot$. This can be seen easily when considering the meaning of φ, as "in all paths, p holds until q holds".

Refinement. If the model checking result of an abstract model is indefinite ($\bot$), a refinement is needed. In the abstraction-refinement framework that we consider here, abstraction is performed by collapsing sets of concrete states into abstract states. Refinement is therefore performed by splitting abstract states, i.e., splitting the sets of concrete states they represent.

When using the coloring algorithms of [9, 10], an indefinite result is accompanied with a *failure state* and a *failure cause*. The failure cause is either a literal whose value in the failure state is $\bot$, or an outgoing may transition of the failure state in the underlying model which is not a must transition. Refinement is then performed by splitting the abstract states in a way that eliminates the failure cause. For example, if the failure cause is a may transition which is

not a must transition, then the source state of the transition is split such that the set of concrete states it represents is divided into states that have a corresponding outgoing transition and states that do not. Typically, the refinement is performed globally, thus further abstract states are split (see [9, 10]).

3. Partial Coloring and Subgraphs

In the following sections we use the game-based model checking in order to identify and focus on the places where the dependencies between components of the system affect the model checking result. In this section we set the basis for this, by investigating properties of the game graph and the coloring algorithms.

Specifically, the coloring algorithms of [9, 10] have the important property that they can be applied on a partially colored graph, in which case they extend the given coloring to the rest of the graph in a correct way. Moreover, the coloring can also be applied on a partially colored *subgraph*, and under certain assumptions it will yield a correct coloring of the subgraph. We now formalize this property.

A partially colored subgraph G' of the game graph G of a 3-valued model checking game induces a new 3-valued parity game. The *induced game* differs from the original one in the omission of the nodes (and edges) outside G'. In addition, the nodes which are colored T by the initial coloring function become terminal nodes in N_1 (i.e., winning for Player 0), the nodes colored F become terminal nodes in N_0 (i.e., winning for Player 1), and the nodes colored ? become terminal nodes in N_{tie}. The priorities remain the same.

Due to their nature, as algorithms for solving a 3-valued parity game, the coloring algorithms can be applied on the induced game. Similarly to before, instead of referring to the algorithms as applied on the induced game, we simply refer to them as applied on G' with an initial coloring function. To formalize the conditions that ensure the correctness of the coloring in this case we need the following definitions.

Definition 3.1. *Let G be the game graph of a 3-valued model checking game and χ_F its correct coloring function. For a non-terminal node n in G we define its* witnessing sons *as follows, depending on its type:*

$\wedge, \Box$: *the witnessing sons are those colored F or ? by χ_F.*

$\vee, \Diamond$: *the witnessing sons are those colored T or ? by χ_F.*

deterministic: *the witnessing son is the only son.*

The sons are witnessing in the sense that they suffice to determine the color (winner) of the node, thus removing the rest of the node's sons from the graph does not damage the result of the coloring. More specifically, when considering the game induced by the graph after the remaining node's sons are removed, the winner (and thus the color) remains unchanged. For example, if a $\wedge$-node or a $\Box$-node has no witnessing sons, meaning all its sons are colored T (i.e., won by Player 0), then the node should be colored T as well, i.e., the winner in it is Player 0. This is also the winner (color) in the induced game when keeping only the witnessing sons (i.e., when no sons remain and the node becomes a terminal node in N_1). Otherwise, the witnessing sons determine whether the node should be colored F or ?, thus one can correctly color the node by considering only them, in the induced game.

Definition 3.2. *A subgraph G' of a game graph G of a 3-valued model checking game is* closed *if every node in G' is either a terminal node, or* all *its witnessing sons (and corresponding edges) from G are also in G'.*

Theorem 3.3. *Consider a closed subgraph G' of a game graph G of a 3-valued model checking game with a partial coloring function χ_I which is correct and defined over (at least) all the terminal nodes in G'. Then applying the coloring algorithm of [9] or [10] on G' with χ_I as an initial coloring results in a correct coloring of G'.*

Proof: It suffices to show that the winner of each node in G' is the same both in the induced game and in the full game. This ensures the correctness of the coloring which is performed by solving the 3-valued parity game.

For the terminal nodes of G' this is clear, since the initial coloring of the terminal nodes is correct (see Definition 2.6). Thus, a terminal node in N_0 is a node that was colored F in the full graph, meaning that Player 1 wins from it in both games. Similarly for the terminal nodes in N_1.

As for the non-terminal nodes in G', we first recall that $\vee$ and $\diamond$ nodes are controlled by Player 0, whereas $\wedge$ and $\Box$ nodes are controlled by Player 1. For $\vee$ and $\diamond$ nodes, controlled by Player 0, the non-witnessing sons are colored F, which means they are winning for Player 1 in the full game, thus Player 0 will not use them in a winning strategy. Dually, for $\wedge$ and $\Box$ nodes, controlled by Player 1, the non-witnessing sons are colored T, which means they are winning for Player 0, thus Player 1 will not use them in a winning strategy.

To show that the winner of each non-terminal node in G' is the same in both games, we show that each winning strategy in the full game translates to a winning strategy of the same player in the induced game, and vice versa. Consider such a node n:

Suppose that Player σ has a winning strategy in the full game starting at n. Then, the same strategy is a winning strategy in the induced game, with the exception that in the "new" terminal nodes of G' the strategy is "pruned", and the winner remains the same due to the above claim regarding the terminal nodes of G'. Note that the winning strategy in the full game never uses sons which are not witnessing (since as explained above, in the nodes controlled by Player σ the non-witnessing sons are not winning for σ). Therefore the strategy is well defined in the induced game as well.

For the opposite direction, suppose that Player σ has a winning strategy in the induced game starting at n. Then the same strategy is a winning strategy in the full game, except that starting from the "new" terminal nodes of G' the "original" strategy is used – again, due to the above claim the winner there remains the same. Moreover, starting from the non-witnessing sons of nodes controlled by Player $1 - \sigma$ (which are not present in the induced game), the original winning strategy of σ is used (as explained above such non-witnessing sons are winning for σ in the full game), and the winner remains the same. □

In fact, for the coloring of the subgraph to be correct, not *all* the witnessing sons are needed, as long as there is enough information to explain the correct coloring of each uncolored node. However, we will see that in our case we will need all of them, as we will deduce from the game graph of one component to the game graph of the full system, where some of the nodes will be removed and for some an indefinite color (?) will change into T or F. This means that some of the witnessing sons will not remain witnessing sons in the game graph of the full system. Thus, we will not be able to know a-priori which of them is the "right" choice to include in a way that will also provide the necessary information for a correct coloring in the game graph of the full system.

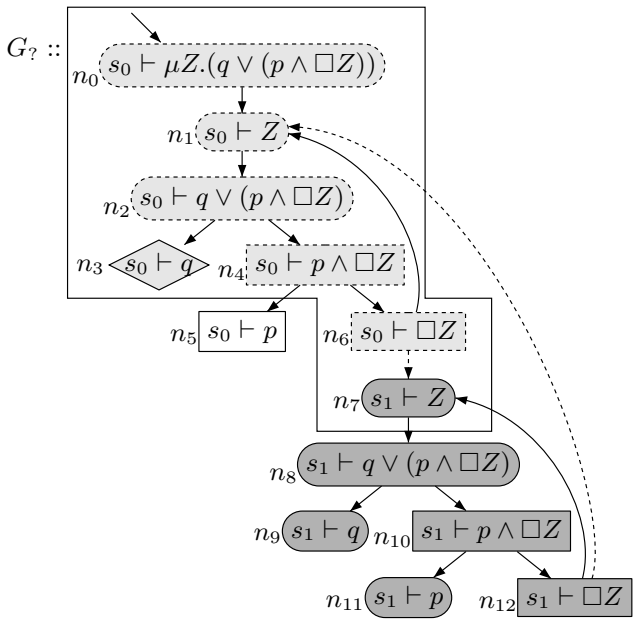


Figure 3: A ?-subgraph (enclosed by a line) as a subgraph of the full game graph. Dashed nodes are uncolored by the initial coloring function of $G_?$.

Another notion that we will need later is the following.

Definition 3.4 (?-Subgraph). *Let G be a colored graph whose initial node is colored ?. The* ?-subgraph *is the least subgraph*[3] *$G_?$ of G that obeys the following:*

- *the initial node is in $G_?$ (and is the initial node of $G_?$).*
- *For each node in $G_?$ which is colored ? in G all its witnessing sons (and corresponding edges) in G are included in $G_?$.*

$G_?$ is accompanied with a partial coloring function χ_I which is defined over the terminal nodes in $G_?$, and colors them as the coloring function χ_F of G.

The ?-subgraph $G_?$ and its initial coloring meet the conditions of Thm. 3.3. Intuitively, this means that $G_?$ contains *all* the information regarding the indefinite result of the initial node.

Example 3.5. Fig. 3 provides an example of the ?-subgraph $G_?$ of the game graph G from Fig. 2(b). The node n_5, which is a son of the $\wedge$-node n_4, is not included in the ?-subgraph since it is not a witnessing son (it is colored T). The node n_7 becomes a terminal node since it is colored by a definite color (F). Its descendants $n_8, \ldots, n_{12}$ are therefore excluded from $G_?$. The figure also depicts the initial coloring of $G_?$, where the terminal nodes n_3 and n_7 are colored as in G, whereas the rest of the nodes are uncolored.

[3] By 'least' subgraph we refer to the smallest subgraph in terms of the number of nodes and edges.

4. Compositional Model Checking

In compositional model checking the goal is to verify a formula φ on a compound system $M_1 \| M_2$. In our setting M_1 and M_2 are Kripke structures that synchronize on the joint labelling of the states. Since a Kripke structure is a special case of a KMTS where $R = R^+ = R^-$, we define the composition for the more general case of KMTSs. In the following we denote by Lit_1 and Lit_2 the sets of literals over AP_1 and AP_2, resp.

Definition 4.1. *Two KMTSs* $M_1 = (AP_1, S_1, s_1^0, R_1^+, R_1^-, L_1)$ *and* $M_2 = (AP_2, S_2, s_2^0, R_2^+, R_2^-, L_2)$ *are* composable *if their initial states agree on their joint labeling, i.e.* $L_1(s_1^0) \cap Lit_2 = L_2(s_2^0) \cap Lit_1$.

Definition 4.2. *Let* $M_1 = (AP_1, S_1, s_1^0, R_1^+, R_1^-, L_1)$ *and* $M_2 = (AP_2, S_2, s_2^0, R_2^+, R_2^-, L_2)$ *be two composable KMTSs. We define their composition, denoted* $M_1 \| M_2$*, to be the KMTS* $(AP, S, s^0, R^+, R^-, L)$*, where*

- $AP = AP_1 \cup AP_2$
- $S = \{(s_1, s_2) \in S_1 \times S_2 \mid L_1(s_1) \cap Lit_2 = L_2(s_2) \cap Lit_1\}$
- $s^0 = (s_1^0, s_2^0)$
- $R^+ = \{((s_1, s_2), (t_1, t_2)) \in S \times S \mid (s_1, t_1) \in R_1^+ \text{ and } (s_2, t_2) \in R_2^+\}$
- $R^- = \{((s_1, s_2), (t_1, t_2)) \in S \times S \mid (s_1, t_1) \in R_1^- \text{ and } (s_2, t_2) \in R_2^-\}$
- $L((s_1, s_2)) = L(s_1) \cup L(s_2)$

In particular, if M_1 *and* M_2 *are Kripke structures with transition relations* R_1 *and* R_2 *resp., then* $M_1 \| M_2$ *is a Kripke structure with* $R = \{((s_1, s_2), (t_1, t_2)) \in S \times S \mid (s_1, t_1) \in R_1 \text{ and } (s_2, t_2) \in R_2\}$.

From now on we fix AP to be $AP_1 \cup AP_2$. For $i \in \{1, 2\}$ we use $\bar{i}$ to denote the remaining index in $\{1, 2\} \setminus \{i\}$.

We use the mechanism produced for abstractions designed to preserve full branching time logics for the purpose of compositional verification. The basic idea is to view each Kripke structure M_i as a partial model that abstracts $M_1 \| M_2$.

Definition 4.3. *Let* $M_i = (AP_i, S_i, s_i^0, R_i, L_i)$ *be a Kripke structure. We lift* M_i *into a KMTS* $M_i\uparrow = (AP, S_i, s_i^0, R_i^+\uparrow, R_i^-\uparrow, L_i\uparrow)$ *over* AP *where* $R_i^+\uparrow = \emptyset$*,* $R_i^-\uparrow = R_i$ *and* $L_i\uparrow(s) = L_i(s)$.

That is, we view M_i as a KMTS $M_i\uparrow$ over AP (rather than AP_i). This immediately makes the value of each literal over $AP \setminus AP_i$ in each state of $M_i\uparrow$ indefinite (as neither p nor $\neg p$ are in $L_i(s)$) – indeed, it depends on $M_{\bar{i}}$. In addition, each transition of M_i is considered a may transition (since in the composition it might be removed if a matching transition does not exist in $M_{\bar{i}}$, but transitions can never be added). Note that $M_1 \| M_2$ can still be completely reconstructed from $M_1\uparrow$ and $M_2\uparrow$.

Theorem 4.4. *For each* $i \in \{1, 2\}$*,* $M_1 \| M_2 \preceq M_i\uparrow$*. The mixed simulation relation* $H \subseteq S \times S_i$ *is given by* $\{((s_1, s_2), s_i) \mid (s_1, s_2) \in S\}$.

Proof: For a state $s = (s_1, s_2)$ of $M_1 \| M_2$ and $i \in \{1, 2\}$ let $\pi_i(s)$ denote the projection of the pair on its ith component. The mixed simulation relation $H \subseteq S \times S_i$ is given by $\{(s, s_i) \mid \pi_i(s) = s_i\}$. That is, a state s_i of $M_i\uparrow$ is related by a mixed simulation relation to all the states of $M_1 \| M_2$ where the corresponding state in the pair is s_i. Clearly, the initial states are 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:

- $L(s) = L_1(\pi_1(s)) \cup L_2(\pi_2(s))$ (by the definition of composition), and the latter is clearly a superset of $L_i(\pi_i(s)) = L_i(s_i)$.
- The requirement for must transitions is met vacuously as there are no must transitions in $M_i\uparrow$.
- Let $(s, s') \in R$. Then by definition of the composition, in particular for i, there exists a transition $(\pi_i(s), \pi_i(s')) \in R_i$, meaning that $(\pi_i(s), \pi_i(s')) \in R_i^-\uparrow$ (by the definition of $M_i\uparrow$). In addition, by the definition of H, $(s', \pi_i(s')) \in H$.

□

Since each $M_i \uparrow$ abstracts $M_1 \| M_2$, we are able to first consider each component separately: Thm. 2.3 ensures that if φ has a definite value (tt or ff) in $M_i\uparrow$ under the 3-valued semantics, then the same value holds in $M_1 \| M_2$ as well. In particular, the values in $M_1\uparrow$ and $M_2\uparrow$ cannot be contradictory, and it suffices that one of them is definite in order to determine the value in $M_1 \| M_2$.

The more typical case is that the value of φ on both $M_1\uparrow$ and $M_2\uparrow$ is indefinite. This reflects the fact that φ depends on both components and their synchronization. Typically, an indefinite result requires some refinement of the abstract model. In our case refinement means considering the composition with the other component. Still, in this case as well, having considered each component separately can guide us into focusing on the places where we indeed need to consider the composition of the components.

The game-based approach to model checking provides a convenient way for presenting this information. If the KMTS $M_i\uparrow$ is model checked using the algorithm of [9] or [10], then the result is a colored game graph, in which T and F represent definite results (i.e. truth values that hold no matter what the environment is), but the ? color needs to be resolved by considering the composition. This is where the ?-subgraph (see Def. 3.4) becomes handy, as it points out the places where this is really needed.

The ?-subgraph for each component is computed top-down, starting from the initial node. As long as a node colored ? is encountered, the search continues in a BFS manner by including the witnessing sons. Definite nodes which are included in the subgraph become terminal nodes, and their coloring defines the initial coloring function.

The ?-subgraphs of the two colored graphs present all the indefinite information that results from the dependencies between the components. Thus, to resolve the indefinite result, we compose the ?-subgraphs.

Definition 4.5 (Product Graph). *Let M_1 and M_2 be two composable Kripke structures, and let G_1 and G_2 be the game graphs of $\varphi \in \mathcal{L}_\mu$ and the lifted KMTSs, $M_1\uparrow$ and $M_2\uparrow$ resp., such that the initial nodes in both graphs are colored ?. Let $G_{?1}$ and $G_{?2}$ be the corresponding ?-subgraphs with initial nodes $s_1^0 \vdash \varphi$ and $s_2^0 \vdash \varphi$ resp. We define the product of $G_{?1}$ and $G_{?2}$ to be the least graph $G_| = (N_|, n_|^0, E_|^+, E_|^-)$ such that:*

- $n_|^0 = (s_1^0, s_2^0) \vdash \varphi$ *is the initial node in* $N_|$.
- *If* $(s_1, s_2) \vdash \psi \in N_|$ *and* $(s_1 \vdash \psi, s_1' \vdash \psi') \in E_1^-$ *and* $(s_2 \vdash \psi, s_2' \vdash \psi') \in E_2^-$ *and* $L_1(s_1') \cap Lit_2 = L_2(s_2') \cap Lit_1$ *(i.e.* (s_1', s_2') *is a state of* $M_1 \| M_2$*), then:* $(s_1', s_2') \vdash \psi' \in N_|$ *and* $((s_1, s_2) \vdash \psi, (s_1', s_2') \vdash \psi')$ *is in* $E_|^+$ *and* $E_|^-$.

Note that all the edges in $G_|$ are must edges, whereas in the ?-subgraphs we had may edges (the transitions of each component were treated as may transitions in the lifted version). This is because the product graph already refers to the complete system $M_1 \| M_2$, where all transitions are concrete transitions (modeled as must transitions).

The product graph is constructed by a top-down traversal of the subgraphs, where, starting from the initial nodes, nodes that share the same formulas and whose states agree on the joint labeling are composed (recall that s_1^0 and s_2^0 agree on their joint labeling). Whenever two non-terminal nodes are composed, the outgoing edges are computed as the product of their outgoing edges, limited to legal nodes (w.r.t. the restriction to states that agree on their labeling). In particular, this means that if a node in one subgraph has no matching node in the other, then it will be omitted from the product graph. In addition, when a terminal node of one subgraph is composed with a non-terminal node of the other, the resulting node is a terminal node in $G_|$.

We accompany $G_|$ with an initial coloring function for its terminal nodes based on the initial coloring functions of the two subgraphs. We use the following observation:

Proposition 4.6. *Let* $n = (s_1, s_2) \vdash \psi$ *be a terminal node in* $G_|$*. Then 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 subgraph, in which case at least one of them is colored by a definite color by the initial coloring of its subgraph, and contradictory definite colors are impossible. We denote this color by* $col(n)$*; Or (b) both* $s_1 \vdash \psi$ *and* $s_2 \vdash \psi$ *are non-terminal nodes but no outgoing edges were left in their composition.*

Proof: Clearly, if $s_1 \vdash \psi$ and $s_2 \vdash \psi$ are both non-terminal nodes, then for $n = (s_1, s_2) \vdash \psi$ to be a terminal node in $G_|$, it has to be the case that no outgoing transitions were left in the composition of $s_1 \vdash \psi$ and $s_2 \vdash \psi$. This refers to case (b).

As for case (a), if at least one of $s_1 \vdash \psi$ and $s_2 \vdash \psi$ is a terminal node in its ?-subgraph, then we show that at least one of them is colored by a definite color: First, by the construction of a ?-subgraph, an indefinite color for a terminal node is only possible when its subformula is a literal. This is because if a node with any other formula is colored ?, then it has at least one witnessing son which will be included in the ?-subgraph, making the node non-terminal. For example, a $\Box$-node which is colored ? has at least one son which is colored $\neq T$, and is thus a witnessing son. A literal only has an indefinite value if it refers to a local atomic proposition of the other component, in which case the node in the other component has a definite color.

In addition, contradictory definite colors cannot exist due to the correctness of the coloring w.r.t. the 3-valued semantics: if $s_1 \vdash \psi$ is colored T, then the value of ψ in s_1 is tt, and by the mixed simulation relation, its value in (s_1, s_2'), for every s_2' that is composable with s_1, is also tt. By the same arguments, if $s_2 \vdash \psi$ is colored F, then the value of ψ in (s_1', s_2), for every s_1' that is composable with s_2, is also ff. This leads to contradiction since s_1 and s_2 are composable. $\square$

Definition 4.7. *We define the initial coloring function* χ_I *of* $G_|$ *as follows. Let n be a terminal node in* $N_|$*. If it fulfills case (a) of Prop. 4.6, then* $\chi_I(n) = col(n)$*. If it fulfills case (b), then* $\chi_I(n) = T$ *if n is a* $\wedge$*-node or a* $\Box$*-node, and* $\chi_I(n) = F$ *if n is a* $\vee$*-node or a* $\Diamond$*-node.* χ_I *is undefined for the rest of the nodes.*

In particular, if a terminal node in $G_|$ results from a terminal node which is colored by ? in one subgraph and a terminal node which is colored by some definite color in the other (case (a)), then the definite color takes over.

Note that a terminal node that fulfills case (b) cannot be deterministic, thus all the possible cases are covered by the definition of the initial coloring function of $G_|$.

Note further that the initial coloring function of the product graph colors all the terminal nodes by definite colors. Along with the property that all the edges in the product graph are must edges, this reflects the fact that the composition resolves all the indefinite information that existed in each component when it was considered separately. Therefore, when applying (one of) the coloring algorithms to the product graph, all the nodes are colored by definite colors (in fact, a 2-valued coloring can be applied).

Theorem 4.8. *The resulting product graph $G_|$ is a closed subgraph of the game graph over $M_1 \| M_2$. In addition, the initial coloring function is correct w.r.t. $M_1 \| M_2$ and defined over all the terminal nodes in the subgraph.*

Proof: We first show that $G_|$ is a closed subgraph of the game graph that corresponds to the composition $M_1 \| M_2$. It is easy to see that it is a subgraph, since the structure in terms of the subformulas and edges is maintained. We now show that the subgraph is closed. Assume to the contrary that it contains a non-terminal node n whose witnessing sons are not all included. Let n' be such a witnessing son. This means that both of the nodes that correspond to n in the ?-subgraphs of the two components are colored by indefinite colors (otherwise, if at least one of them was colored by a definite color, it would have been a terminal node and so would n). Moreover, at least one of the nodes that correspond to the witnessing son n' is not included in its subgraph (otherwise n' would have been included in the product graph). Denote this node by $\widetilde{n'}$. By the correctness of the 3-valued semantics, the color of $\widetilde{n'}$ in its game graph is either the same as the color of n' in the game graph for the composition $M_1 \| M_2$, or indefinite. However, in both cases this makes it a witnessing son in its game graph, which means it should have been included in the ?-subgraph. To see why $\widetilde{n'}$ is a witnessing son of $\widetilde{n}$ (the node that corresponds to n) in its game graph, first note that if $\widetilde{n'}$ is colored ?, then this is immediate. If $\widetilde{n'}$ is colored by a definite color in its game graph, then, as explained above, its color is the same as the color of n' in the full game graph of $M_1 \| M_2$. Therefore, since n' is a witnessing son of n, and since the types of n and $\widetilde{n}$ are the same, it holds that $\widetilde{n'}$ is also a witnessing son of $\widetilde{n}$.

We show that the initial coloring function is correct by a case analysis. For terminal nodes that are colored based on case (a) this directly results from the correctness of the 3-valued semantics (Theorem 2.3). As for terminal nodes that are colored based on case (b), consider the case of a node n with a $\wedge$ or a $\Box$ formula (the other case is dual). If n is also a terminal node in the full game graph of $M_1 \| M_2$, then it must consist of a $\Box$ formula and of a state of $M_1 \| M_2$ that has no successors, which means it satisfies the $\Box$ formula and the T-color of n is correct. Otherwise, since all its sons were removed during the composition, this means that every one of them corresponds to a node that does not exist in the ?-subgraph of at least one component, which means it was colored T in the game graph of the component. Again, the preservation theorem ensures that the color of all the sons in the game graph of $M_1 \| M_2$ is also T, and thus T is the correct color of n. □

By Thm. 3.3, this means that coloring $G_|$ results in a correct result w.r.t. the model checking of φ in $M_1 \| M_2$. Thus, to model check φ on $M_1 \| M_2$ it remains to color $G_|$. Note that the full graph

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

1. Construct the game graph G_i for φ and $M_i\uparrow$.
2. Apply the 3-valued coloring on G_i. Let χ_i be the resulting coloring 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 ?-subgraphs for G_1 and G_2.
2. Construct the product graph $G_\|$ of the ?-subgraphs.
3. Apply the 3-valued coloring on $G_\|$ (with the initial coloring function).

Return the model checking result corresponding to $\chi_\|(n_\|^0)$.

Figure 4: Compositional model checking algorithm.

for $M_1 \| M_2$ is not constructed. The resulting compositional model checking algorithm appears in Fig. 4.

Example 4.9. Consider the components depicted in Fig. 5(a). The atomic proposition o (short for *output*) is local to M_1, i (*input*) is local to M_2, and r (*receive*) is the only joint atomic proposition that M_1 and M_2 synchronize on. Suppose we wish to verify in $M_1 \| M_2$ the property $\Box(\neg i \vee \Diamond o)$, which states that in all the successor states of the initial state, an *input* signal implies that there is a successor state where the *output* signal holds. Fig. 5(b) depicts the colored game graph of each (lifted) component, and highlights the ?-subgraph of each of them. The product graph and its coloring is depicted in Fig. 5(c), as an "intersection" of the two subgraphs. All the edges in the product graph are must edges. All nodes, and in particular the initial node, are colored T, thus the property is verified. One can see that most of the efforts were done on each component separately, and the product graph only considers a small part of the compound system.

5. Adding Abstraction

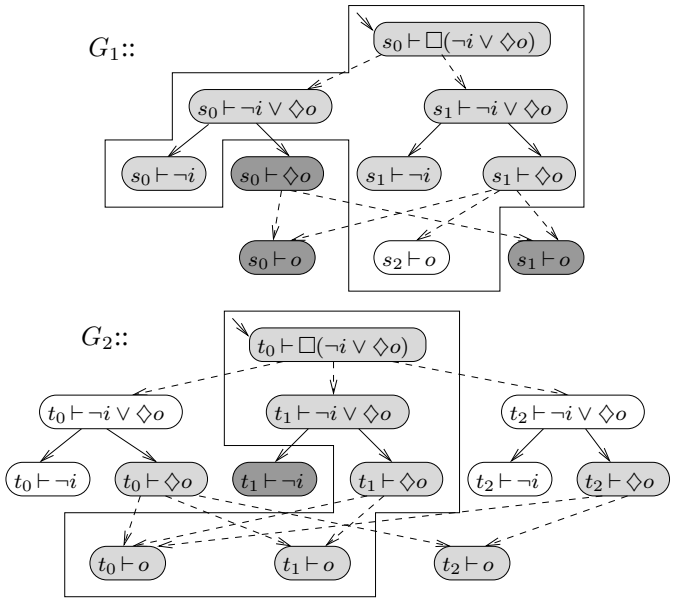
In Section 4 we considered concrete components. The indefinite results on each component resulted only from their interaction, and were resolved by composing the indefinite parts. We now combine this idea with existing abstraction-refinement techniques.

5.1. Motivation

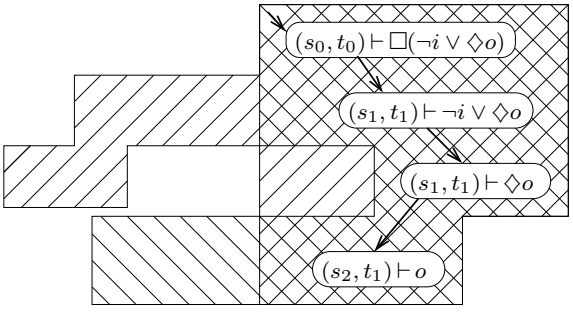
Even when using the compositional approach, the resulting game graphs might still be too big to handle since the components themselves might still be very big, possibly infinite. To further reduce the size of the checked components, we combine our compositional approach with abstraction. Abstraction not only reduces the state-space of the components, but also allows to handle infinite-state components by abstracting them into finite-state components.

It turns out that abstraction has another benefit in our context. Namely, composing the ?-subgraphs of two components, as suggested in Section 4, corresponds to refining *all* possible

(a)



(b)



(c)

Figure 5: (a) Components, (b) their game graphs and their ?-subgraphs (enclosed by a line), and (c) the product graph.

failure causes. We now show how to use abstraction in order to make the refinement more local and gradual by eliminating *one* failure cause at a time.

Suppose that the coloring of the game-graph G_1 for the lifted concrete component $M_1\uparrow$ results in an indefinite result. We wish to eliminate the failure cause returned by the coloring algorithm for $M_1\uparrow$. Suppose that s is the failure state. It abstracts all the states of $M_1\|M_2$ that consist of s and a matching state of M_2. Eliminating the cause for failure amounts to exposing from M_2 the information that involves the failure, and splitting s accordingly. For example, in Fig. 5, a possible failure cause in G_1 is the may transition of $M_1\uparrow$ from s_1 to s_2. In order to either remove it or turn it into a must transition, we need to consider all the states of M_2 which are composable with s_1. These are the states labeled r. We need to find out which of them have a transition to a state labeled r (i.e., a state composable with s_2), and which of them do not.

Clearly, the complete composition of the ?-subgraphs achieves this goal. However, it exposes more information than relevant for the given failure cause: It exposes *all* the information relevant to any possible cause for failure. In the general case, however, eliminating all failure causes is not necessary. Thus we do not want to resort to that. We now sketch the idea that allows us to only consider the information from M_2 that is needed for eliminating the particular failure cause of $M_1\uparrow$. This will be described more formally in Section 5.2.

We abstract M_2 into $\hat{M}_2$. We start with a coarsest abstraction of M_2 w.r.t. $AP_1 \cap AP_2$, where each state is abstracted by its labeling, restricted to $AP_1 \cap AP_2$. This can be viewed as an abstraction that collapses all states that agree on $AP_1 \cap AP_2$ into a single abstract state.

Definition 5.1. *Let $M_i = (AP_i, S_i, s_i^0, R_i, L_i)$ be a Kripke structure. The* coarsest abstraction *for M_i w.r.t. $AP' \subseteq AP_i$ is the KMTS $\hat{M}_i^* = (AP_i, 2^{AP'}, L_i(s_i^0) \cap AP', \emptyset, 2^{AP'} \times 2^{AP'}, L_i^*)$, where for $\hat{s} \in 2^{AP'}$, $L_i^*(\hat{s}) = \hat{s} \cup \{\neg p \mid p \in AP' \setminus \hat{s}\}$.*

Theorem 5.2. *$M_i \preceq \hat{M}_i^*$. The mixed simulation is $\{(s_i, L_i(s_i) \cap AP') \mid s_i \in S_i\}$.*

Proof: We show that $(s_i, L_i(s_i) \cap AP') \mid s_i \in S_i\}$ is a mixed simulation relation from M_i to $\hat{M}_i^*$. Clearly, the initial states are in H since the initial state of $\hat{M}_i^*$ is $L_i(s_i^0) \cap AP'$.

Let $(s_i, \hat{s}) \in H$, i.e. $\hat{s} = L_i(s_i) \cap AP'$. Then:

- $L_i(s_i) \supseteq L_i(s_i) \cap Lit' = L_i^*(\hat{s})$. The latter equality holds since by the definition of the coarsest abstraction, $L_i^*(\hat{s}) = \hat{s} \cup \{\neg p \mid p \in AP' \setminus \hat{s}\}$. $\hat{s} = L_i(s_i) \cap AP'$. Thus, $\{\neg p \mid p \in AP' \setminus \hat{s}\} = \{\neg p \mid p \in AP' \setminus (L_i(s_i) \cap AP')\} = \{\neg p \mid p \in AP'$ and $p \notin L_i(s_i)\} = \{\neg p \mid p \in AP'$ and $\neg p \in L_i(s_i)\} = L_i(s_i) \cap \{\neg p \mid p \in AP'\}$. Therefore, altogether, the union of $\hat{s}$ and $\{\neg p \mid p \in AP' \setminus \hat{s}\}$ is equal to $L_i(s_i) \cap Lit'$.

- The requirement for must transitions is met vacuously as there are no must transitions in $\hat{M}_i^*$.

- The requirement for may transitions is also met trivially as every pair of states in $\hat{M}_i^*$ have a may transition between them.

□

An example of the coarsest abstraction of M_2 from Fig. 5(a) w.r.t. $\{r\}$ appears in Fig. 6. The construction of the coarsest abstraction requires almost no knowledge of the component. More precise transitions can be computed as in [8]. Starting from the coarsest abstraction of M_2, we iteratively model check the composition of M_1 and the abstract model $\hat{M}_2$. The model checking is performed in a compositional fashion, similarly to Section 4, without computing the full

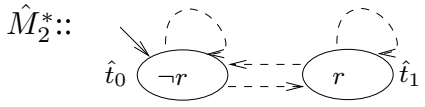


Figure 6: Coarsest abstraction of M_2 from Fig. 5(a) w.r.t. $\{r\}$.

composition. If the result in some iteration is indefinite, we refine $\hat{M}_2$ depending on the failure cause over $M_1 \| \hat{M}_2$. Recall that our purpose was to eliminate a failure cause over $M_1\uparrow$. Since we start with a coarsest abstraction of M_2 w.r.t. the joint atomic propositions, $M_1 \| \hat{M}_2$ is initially isomorphic to $M_1\uparrow$. As a result, in the first iteration the failure cause over $M_1 \| \hat{M}_2$ reflects the failure cause over $M_1\uparrow$, and the refinement of $\hat{M}_2$ indeed exposes the relevant information from M_2. Similarly, in the next iterations, the failure cause over $M_1 \| \hat{M}_2$ reflects the failure cause over $M_1\uparrow$, after taking into consideration the elimination of previous failure causes. In this sense, in each iteration we eliminate one failure cause over $M_1\uparrow$, and $\hat{M}_2$ "accumulates" the information required to eliminate these failure causes.

This means that we keep one of the components, M_1, concrete, and construct an abstract environment for it, by applying an iterative abstraction-refinement on M_2, where refinement is aimed at eliminating the indefinite results that arise when considering M_1 with the abstract environment. This approach is reminiscent of an asymmetric Assume-Guarantee rule, in which to verify that $M_1 \| M_2$ satisfies φ, one comes up with an assumption A on M_2 and shows that (1) when M_1 is composed with A it satisfies φ, and in addition that (2) M_2 satisfies the assumption A. In our setting, the second premise holds automatically since we use a conservative abstraction of M_2 as an assumption. The next step is to make the approach symmetric by abstracting both components. This amounts to constructing abstract environments for both the components. In this case, refinement also needs to be applied on both components.

5.2. Compositional Abstraction-Refinement

We now describe in detail the combination of the compositional approach with abstraction-refinement. This provides a framework for using both the asymmetric and the symmetric approaches sketched above. On the one hand, we enhance the compositional model checking of Section 4 by using abstraction and a more gradual refinement. On the other hand, we enhance the abstraction-refinement framework by making both the abstract model checking and the refinement compositional. We no longer require that the state spaces of the concrete components are finite, as long as the abstract state spaces are.

Compositional Abstraction. Composition of abstract models (KMTSs) is defined in Def. 4.2. In order to ensure that the composition of two abstract models $\hat{M}_1 = (AP_1, \hat{S}_1, \hat{s}_1^0, R_1^+, R_1^-, \hat{L}_1)$ and $\hat{M}_2 = (AP_2, \hat{S}_2, \hat{s}_2^0, R_2^+, R_2^-, \hat{L}_2)$, for M_1 and M_2 respectively, results in an abstract model for $M_1 \| M_2$, we consider *appropriate* abstract models w.r.t. $AP_1 \cap AP_2$. We say that $\hat{M}_i$ is an *appropriate* abstract model of M_i w.r.t. $AP_1 \cap AP_2$ if $\hat{M}_i$ and M_i are related by a mixed simulation relation which is appropriate w.r.t. $AP_1 \cap AP_2$, as defined below.

Definition 5.3. *Let $H \subseteq S_i \times \hat{S}_i$ be a mixed simulation from M_i to $\hat{M}_i$, both defined over AP_i. We say that H is* appropriate *w.r.t. $AP' \subseteq AP_i$ if for every $(s_i, \hat{s}_i) \in H$, $L_i(s_i) \cap Lit' = \hat{L}_i(\hat{s}_i) \cap Lit'$, where Lit' denotes the set of literals over AP'.*

In particular, the coarsest abstraction w.r.t. $AP_1 \cap AP_2$ (see Def. 5.1) is appropriate w.r.t. $AP_1 \cap AP_2$. Appropriateness of $\hat{M}_1$ and $\hat{M}_2$ w.r.t. $AP_1 \cap AP_2$ means that the abstraction of each component only identifies states that agree on their labelings w.r.t. the joint atomic propositions. It ensures that if $(\hat{s}_1, \hat{s}_2)$ is a state of the abstract composition and $\hat{s}_1$ abstracts s_1 and $\hat{s}_2$ abstracts s_2, then since $\hat{s}_1$ and $\hat{s}_2$ agree on the joint labeling, then so do s_1 and s_2. This ensures that (s_1, s_2) is a state of the concrete composition, abstracted by $(\hat{s}_1, \hat{s}_2)$. We now have the following.

Theorem 5.4. *Let $\hat{M}_i$ be an appropriate abstract model for M_i w.r.t. $AP_1 \cap AP_2$. Then $M_1 \| M_2 \preceq \hat{M}_1 \| \hat{M}_2$.*

Proof: Let $H_1 \subseteq S_1 \times \hat{S}_1$ and $H_2 \subseteq S_2 \times \hat{S}_2$ denote the mixed simulations between each component and its abstract model. Then the mixed simulation relation $H \subseteq S \times \hat{S}$ is given by $\{(s, \hat{s}) \mid (\pi_1(s), \pi_1(\hat{s})) \in H_1 \text{ and } (\pi_2(s), \pi_2(\hat{s})) \in H_2\}$. Clearly, the pair of initial states $((s_1^0, s_2^0), (\hat{s}_1^0, \hat{s}_2^0))$ is in H because $(s_1^0, \hat{s}_1^0)$ is in H_1 and $(s_2^0, \hat{s}_2^0)$ is in H_2 (since H_1 and H_2 are mixed simulations). Let $(s, \hat{s}) \in H$. Then

- $L(s) = L_1(\pi_1(s)) \cup L_2(\pi_2(s)) \supseteq \hat{L}_1(\pi_1(\hat{s})) \cup \hat{L}_2(\pi_2(\hat{s})) = \hat{L}(s)$ (the inclusion follows since H_1 and H_2 are mixed simulations).
- Let $(\hat{s}, \hat{s}') \in R^+$ in $\hat{M}_1 \| \hat{M}_2$. By the definition of (abstract) composition this implies that for each i, $(\pi_i(\hat{s}), \pi_i(\hat{s}')) \in R_i^+$. Then since H_1 and H_2 are mixed simulations, we conclude that for each i, there exists $t_i \in S_i$ such that $(\pi_i(s), t_i) \in R_i$ and $(t_i, \pi_i(\hat{s}')) \in H_i$. Since $\hat{M}_i$ is an appropriate abstract model for M_i w.r.t. $AP_1 \cap AP_2$, we have that $\hat{L}_i(\pi_i(\hat{s}')) \cap Lit_1 \cap Lit_2 = L_i(t_i) \cap Lit_1 \cap Lit_2$. Moreover, since $(\pi_1(\hat{s}'), \pi_2(\hat{s}'))$ is a state in $\hat{M}_1 \| \hat{M}_2$, then $\hat{L}_1(\pi_1(\hat{s}'))$ and $\hat{L}_2(\pi_2(\hat{s}'))$ agree on the joint atomic propositions, thus so do $L_1(t_1)$ and $L_2(t_2)$, and there exists a state (t_1, t_2) in $M_1 \| M_2$. In addition, by definition of H, $((t_1, t_2), \hat{s}') \in H$. It remains to show that $(s, (t_1, t_2)) \in R$. This is immediate from the definition of the (concrete) composition.
- Let $(s, s') \in R$. Then by definition of the (concrete) composition, there exists a transition $(\pi_i(s), \pi_i(s')) \in R_i$, and since H_i are mixed simulations, there exist $\hat{t}_i$ such that $(\pi_i(\hat{s}'), \hat{t}_i) \in R_i^-$ and $(\pi_i(s'), \hat{t}_i) \in H_i$. Since $\hat{M}_i$ is an appropriate abstract model for M_i w.r.t. $AP_1 \cap AP_2$, we have that $\hat{L}_i(\pi_i(\hat{s}')) \cap Lit_1 \cap Lit_2 = L_i(t_i) \cap Lit_1 \cap Lit_2$. Moreover, since $(\pi_1(s'), \pi_2(s'))$ is a state, then $L_1(\pi_1(s'))$ agrees with $L_2(\pi_2(s'))$ on the joint atomic propositions, thus so do $\hat{L}_1(\hat{t}_1)$ and $\hat{L}_2(\hat{t}_2)$, and there exists a state $(\hat{t}_1, \hat{t}_2)$. In addition, by definition of H, $(s', (\hat{t}_1, \hat{t}_2)) \in H$. It remains to show that $(\hat{s}, (\hat{t}_1, \hat{t}_2)) \in R^-$. This is immediate from the definition of the (abstract) composition.

□

Thus, if each of M_1 and M_2 is abstracted separately by an appropriate abstraction w.r.t. $AP_1 \cap AP_2$, then the composition of the corresponding abstract components $\hat{M}_1$ and $\hat{M}_2$ results in an abstract model for $M_1 \| M_2$. However, we do not wish to construct $\hat{M}_1 \| \hat{M}_2$ and model check it. Instead, we suggest to model check $\hat{M}_1 \| \hat{M}_2$ compositionally.

Compositional (abstract) Model Checking. The general scheme is similar to the concrete case: we first try to make the most out of each (abstract) component separately, and if this does not result in a definite answer, we consider the product of the ?-subgraphs which enable to exchange information via a compact representation. We start by viewing each abstract component $\hat{M}_i$ as a partial model that abstracts their composition $\hat{M}_1 \| \hat{M}_2$.

Definition 5.5. *Let* $\hat{M}_i = (AP_i, \hat{S}_i, \hat{s}_i^0, R_i^+, R_i^-, \hat{L}_i)$ *be a KMTS. We lift* $\hat{M}_i$ *into a KMTS* $\hat{M}_i\uparrow = (AP, \hat{S}_i, \hat{s}_i^0, R_i^+\uparrow, R_i^-\uparrow, \hat{L}_i\uparrow)$ *over AP where* $R_i^+\uparrow = \emptyset$, $R_i^-\uparrow = R_i^-$ *and* $\hat{L}_i\uparrow(\hat{s}) = \hat{L}_i(\hat{s})$.

That is, when $\hat{M}_i$ is lifted into $\hat{M}_i\uparrow$, only the may transitions of $\hat{M}_i$ are useful, because must transitions are not really must w.r.t. $\hat{M}_1 \| \hat{M}_2$. Similarly to the concrete case:

Theorem 5.6. $\hat{M}_1 \| \hat{M}_2 \preceq \hat{M}_i\uparrow$.

Proof: Similar to the proof of Theorem 4.4. □

Corollary 5.7. *If* $\hat{M}_i$ *is an appropriate abstract model for* M_i *w.r.t.* $AP_1 \cap AP_2$, *then* $M_1 \| M_2 \preceq \hat{M}_i\uparrow$.

Proof: Follows immediately from Theorem 5.4 and Theorem 5.6 by the transitivity of $\preceq$. □

Therefore one can model check each of $\hat{M}_i\uparrow$ separately, and the definite results follow through to $M_1 \| M_2$. In fact, it is possible to show that $M_1 \| M_2 \preceq \hat{M}_i\uparrow$ holds even if we omit the appropriateness requirement. Thus appropriateness is not needed for this step. However, it is needed for the next steps, where we deduce from $\hat{M}_1 \| \hat{M}_2$ to $M_1 \| M_2$.

If both checks result in indefinite results, the (abstract) ?-subgraphs for both game graphs are produced and their product is considered. Having composed the ?-subgraphs of the two components resolves dependencies between them, but the result is still abstract, as it refers to the *abstract* composition $\hat{M}_1 \| \hat{M}_2$. This results in two differences compared to the concrete case.

First, the may edges do not necessarily become must edges. Instead, the distinction between may and must edges is determined by the type of the underlying transitions in the (unlifted) abstract models $\hat{M}_i$, which have been ignored so far. Second, it is now possible that a terminal node $n = (\hat{s}_1, \hat{s}_2) \vdash \psi$ in $G_|$ with $\psi = l$ for a local literal $l \in Lit \setminus (Lit_1 \cap Lit_2)$ results from terminal nodes $\hat{s}_1 \vdash l$ and $\hat{s}_2 \vdash l$ which are *both* colored by ? in their subgraphs (one, since l is local to the other component, and is thus treated as indefinite, and the other due to the abstraction). We add this possibility as case (c) to Prop. 4.6 which characterizes the terminal nodes in the product graph $G_|$. It is taken into account when determining the initial coloring of $G_|$.

Definition 5.8 (Abstract Product Graph). *Let* $\hat{M}_1$ *and* $\hat{M}_2$ *be two composable (abstract) KMTSs, and let* G_1 *and* G_2 *be the game graphs of* $\varphi \in \mathcal{L}_\mu$ *and the lifted KMTSs,* $\hat{M}_1\uparrow$ *and* $\hat{M}_2\uparrow$ *resp., such that the initial nodes in both graphs are colored ?. Let* $G_{?1}$ *and* $G_{?2}$ *be the corresponding (abstract) ?-subgraphs. The product graph* $G_| = (N_|, n_|^0, E_|^+, E_|^-)$ *of* $G_{?1}$ *and* $G_{?2}$ *is defined as before (see Def. 4.5), except for the definition of* $E_|^+$*: an edge* $((\hat{s}_1, \hat{s}_2) \vdash \psi, (\hat{s}'_1, \hat{s}'_2) \vdash \psi')$ *in* $E_|^-$ *is also in* $E_|^+$ *iff* $\hat{s}_i R_i^+ \hat{s}'_i$ *for each* $i \in \{1, 2\}$. *The initial coloring function is defined as before (see Def. 4.7), with the addition that a terminal node that fulfills case (c) in the adapted version of Prop. 4.6 is colored ?.*

Theorem 5.9. *The resulting abstract product graph* $G_|$ *is a closed subgraph of the game graph over* $\hat{M}_1 \| \hat{M}_2$. *In addition, the initial coloring function is correct and defined over all the terminal nodes in the subgraph.*

Proof: Similar to the proof of Theorem 4.8, with the following additions. First, in □ and ◇ nodes, the types of the outgoing edges (may vs. must) are determined by the same guidelines as

the types of the transitions in $\hat{M}_1 \| \hat{M}_2$, which ensures that the abstract product graph is a subgraph of the game graph that corresponds to $\hat{M}_1 \| \hat{M}_2$. Second, for terminal nodes that are colored based on case (c) the correctness of the initial coloring function results from the fact that case (c) represents the case of a node $(\hat{s}_1, \hat{s}_2) \vdash l$ in $G_|$, where $l \in Lit_i$ is local to $\hat{M}_i$ but its value in $\hat{s}_i$ is indefinite. This implies that its value in the corresponding state $(\hat{s}_1, \hat{s}_2)$ of $\hat{M}_1 \| \hat{M}_2$ is also indefinite (by the definition of composition), which makes the initial coloring correct. □

Along with Thm. 3.3, this implies that $G_|$ can be colored correctly (w.r.t. the model checking of φ on $\hat{M}_1 \| \hat{M}_2$) using the 3-valued algorithm. If the initial node is colored by a definite color, then by Thm. 5.4 the result holds in $M_1 \| M_2$ as well and we are done.

Compositional Refinement. Since an abstraction is used, the result of the model checking can be $\perp$, in which case the coloring of [9, 10] returns a failure cause that needs to be eliminated. The failure cause is either a literal whose value in a certain state is $\perp$, or a may transition of the underlying model which is not a must transition.

In our setting, the refinement step is done compositionally: If the failure cause is a literal l whose value in the failure state of $\hat{M}_1 \| \hat{M}_2$ is $\perp$, then l has to be a local literal of one of the components. This is because the abstraction is appropriate w.r.t. $AP_1 \cap AP_2$, which implies that no indefinite values for the joint atomic propositions occur in $\hat{M}_1 \| \hat{M}_2$. Thus, refinement need only be applied on the corresponding component.

Otherwise, the failure cause returned by the coloring is a may transition (which is not a must transition) of $\hat{M}_1 \| \hat{M}_2$ that needs to be refined in order to result in a must transition or no transition at all. Let $((\hat{s}_1, \hat{s}_2), (\hat{s}'_1, \hat{s}'_2))$ be this may transition. Then it results from may transitions $(\hat{s}_1, \hat{s}'_1)$ and $(\hat{s}_2, \hat{s}'_2)$ of $\hat{M}_1$ and $\hat{M}_2$ resp., such that at least one of them is not a must transition. In order to refine $((\hat{s}_1, \hat{s}_2), (\hat{s}'_1, \hat{s}'_2))$, one needs to refine the individual may transitions in each component separately. If both of them are not must transitions, then refinement should be applied in each component. This is because a must transition in the composition results from must transitions in *both* components[4]. Otherwise, refinement should only be applied in the component where it is not a must transition.

In each component where refinement is necessary, the refinement can be done as in [8–10]. Moreover, in each component we adopt the incremental approach of [8–10] and avoid unnecessary refinement. In this approach, only nodes with indefinite colors are refined. In our setting, this corresponds to the ?-subgraph of each component. The result is the compositional abstraction-refinement loop presented in Fig. 7.

Note that the must transitions of each abstract component are only used when $G_|$ is constructed. Thus, their computation can be deferred to step 2 and be limited to must transitions that are needed during model checking. Hyper-transitions, pointing to a *set* of states rather than to a single state, can also be used, e.g. with the algorithm of [32].

Using the compositional abstraction-refinement starting from the coarsest abstraction w.r.t. $AP_1 \cap AP_2$ of one or both of the components results in the asymmetric, resp. symmetric, approach described in Section 5.1.

[4] As an optimization, it is possible to refine only one of the components first. If the failure may transition results in no transition at all in one of the components after refinement, then the failure may transition will be removed from the composition as well, regardless of its existence in the other component. Thus, refinement of the other component can be avoided.

Step 0 For $i \in \{1,2\}$, abstract M_i into $\hat{M}_i$ appropriately w.r.t. $AP_1 \cap AP_2$ (e.g. as in Def.5.1).

Step 1 Model check each $\hat{M}_i\uparrow$ separately (for $i \in \{1,2\}$):

1. Construct the game graph G_i for φ and $\hat{M}_i\uparrow$.
2. Apply the 3-valued coloring on G_i. Let χ_i be the resulting coloring 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 $\hat{M}_1 \| \hat{M}_2$:

1. Construct the ?-subgraphs for G_1 and G_2.
2. Construct the (abstract) product graph $G_|$ of the ?-subgraphs.
3. Apply the 3-valued coloring on $G_|$ (with the initial coloring function).

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

Step 3 Refine: Consider the failure cause returned by the coloring of $G_|$ (where $\chi_|(n_|^0) =?$).

If it is $l \in Lit_i$ then refine $\hat{M}_i$; Else let it be the may transition $((\hat{s}_1, \hat{s}_2), (\hat{s}'_1, \hat{s}'_2))$. Then:

1. If $(\hat{s}_1, \hat{s}'_1)$ is not a must transition in $\hat{M}_1$, refine $\hat{M}_1$.
2. If $(\hat{s}_2, \hat{s}'_2)$ is not a must transition in $\hat{M}_2$, refine $\hat{M}_2$.

Refine the ?-subgraphs of G_1 and G_2 accordingly (as in the incremental approach);
Go to Step 1(2) with the refined subgraphs.

Figure 7: Compositional abstraction-refinement algorithm.

Theorem 5.10. *For finite concrete components, iterating the compositional abstraction-refinement process is guaranteed to terminate with a definite answer.*

Example 5.11. We demonstrate the compositional abstraction-refinement scheme on Example 4.9, which also served as a motivating example for introducing abstraction (see Section 5.1). We use the asymmetric approach, where only one component, in our case M_2, is abstracted. Initially, M_2 is abstracted using the coarsest abstraction w.r.t. the only shared atomic proposition r, as depicted in Fig. 6. Namely, $\hat{M}_2 = \hat{M}_2^*$. The (initial) state $\hat{t}_0$ of the coarsest abstraction, labeled $\neg r$, represents the initial concrete state t_0 which is also labeled $\neg r$, and the state $\hat{t}_1$, labeled r, represents the two concrete states t_1 and t_2 which are labeled r. $\hat{M}_2$ is lifted into $\hat{M}_2\uparrow$ and model checked. The game graph G_2 of $\hat{M}_2\uparrow$ is depicted in Fig. 8(a), where the initial node, as well as all other nodes, are colored ?. This is expected since the coarsest abstraction basically reveals no information on the concrete component.

Since all nodes are colored ?, the ?-subgraph of G_2 consists of the entire game graph. Recall that the initial node of the game graph G_1 that corresponds to $M_1\uparrow$ is also colored ? (see Fig. 5(b)). Therefore, the product graph $G_|$ of the ?-subgraphs of G_1 (see Fig. 5(b)) and G_2 is constructed, as depicted in Fig. 8(b). Note that $G_|$ is still abstract: it contains may edges and terminal nodes which are colored ?. In fact, it is isomorphic to the ?-subgraph of G_1. Thus, its coloring is

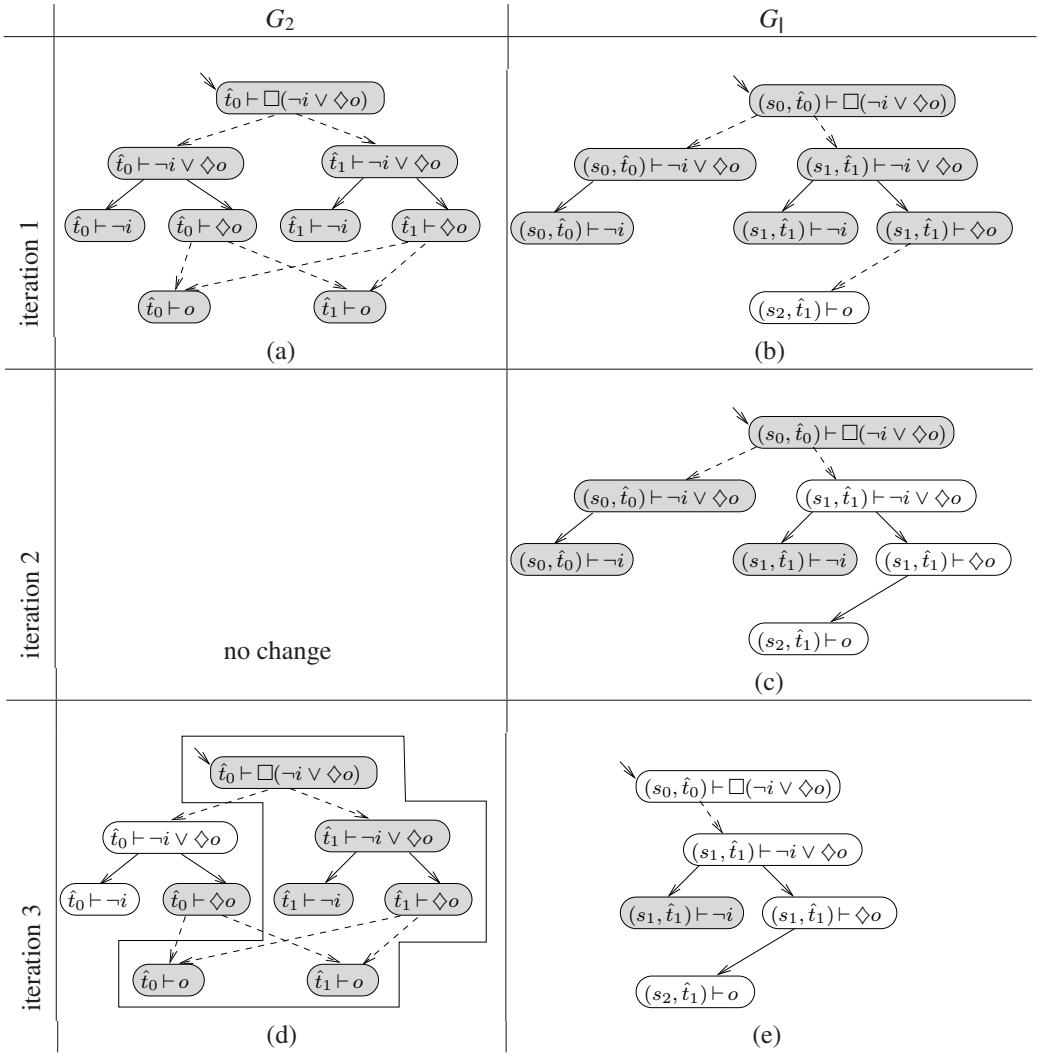


Figure 8: Game graphs arising during the run of the compositional abstraction-refinement algorithm described in Example 5.11.

identical to that of G_1. In particular, it results in an indefinite value, accompanied with failure information.

Suppose that the failure state returned by the coloring is $(s_1, \hat{t}_1)$ with the failure cause being its outgoing may transition to the state $(s_2, \hat{t}_1)$, which is the underlying transition of the may edge from $(s_1, \hat{t}_1) \vdash \Diamond o$ to $(s_2, \hat{t}_1) \vdash o$ in the product graph. This may transition results from the concrete transition from s_1 to s_2 in M_1, and the self may loop of $\hat{t}_1$ in $\hat{M}_2$. Therefore, refinement is needed (only) in $\hat{M}_2$, where the corresponding transition is not a must transition. Note that while the refinement is performed in $\hat{M}_2$, it is in fact based on the point where more information is needed by the model checking instance of M_1, namely, whether or not the transition from s_1 to s_2 has a corresponding transition in M_2. This is because the failure cause is derived from the product graph which is isomorphic to the ?-subgraph of G_1. In particular, the may edge from $(s_1, \hat{t}_1) \vdash \Diamond o$ to $(s_2, \hat{t}_1) \vdash o$ in the product graph, from which the failure cause is derived, reflects the may edge from $s_1 \vdash \Diamond o$ to $s_2 \vdash o$ in G_1, which is one of the failure causes in G_1.

The refinement of $\hat{M}_2$ is aimed at splitting $\hat{t}_1$ such that each of the resulting substates either has a corresponding must transition or no transition at all. However, it turns out that in this case both of the concrete states represented by $\hat{t}_1$ have a corresponding transition, which means that the self loop of $\hat{t}_1$ can simply be added as a must transition and no split is required.

The second iteration starts from the refined ?-subgraph of G_2 (which is the entire graph in this case). In fact, since no states were split, and since in the lifted component all transitions are viewed as may transitions, the so-called refined ?-subgraph remains unchanged and no recoloring is needed. In particular, the ?-subgraph computed in the second iteration remains the same, i.e. it consists of the entire game graph. However, when the product graph is constructed, the may edge from $(s_1, \hat{t}_1) \vdash \Diamond o$ to $(s_2, \hat{t}_1) \vdash o$ becomes a must edge, since it results from the concrete transition from s_1 to s_2 in M_1, and the self loop of $\hat{t}_1$ in $\hat{M}_2$ which turns out to be a must transition. As a result, the coloring changes as depicted in Fig. 8(c). The initial node is still colored ?, and new failure information is provided.

Suppose that the failure state returned by the coloring is now $(s_0, \hat{t}_0)$ with the failure cause being the literal $\neg i$, whose value in $(s_0, \hat{t}_0)$ is indefinite (this failure information is derived from the terminal node $(s_0, \hat{t}_0) \vdash \neg i$ which is colored ? in the product graph). The literal $\neg i$ is local to M_2. Therefore, refinement is again needed in $\hat{M}_2$ only, and it reflects the fact that the information regarding $\neg i$ is needed by the model checking instance of M_1 (to resolve the ?-color of the node $s_0 \vdash \neg i$, which is one of the failure causes in G_1).

The refinement of $\hat{M}_2$ is aimed at splitting $\hat{t}_0$ such that each of the resulting substates is labeled by i or $\neg i$. However, it turns out that in this case the (only) concrete state represented by $\hat{t}_0$ is labeled $\neg i$, which means that the labeling of $\hat{t}_0$ can be updated and no split is required.

The third iteration starts from the refined ?-subgraph of G_2 from the second iteration. Recall that the ?-subgraph in the second iteration consists of the entire game graph. The only change in the refined ?-subgraph is that the terminal node $\hat{t}_0 \vdash \neg i$ is now colored T. As a result, the coloring of the refined ?-subgraph of G_2 changes as described in Fig. 8(d), which also highlights the new ?-subgraph. Fig. 8(e) presents the resulting product graph, and its coloring, where the initial node is now colored T, meaning that the property is verified.

While this small example does not really demonstrate the full power of abstraction, it nicely shows how the use of abstraction achieves the goal of a gradual refinement, where one failure cause is eliminated at a time. In this example, each refinement step reveals from M_2 additional information based on one failure cause found in the product graph, which reflects the model checking instance of M_1. This information is accumulated in $\hat{M}_2$.

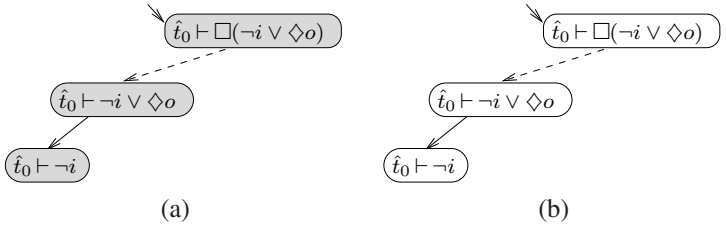


Figure 9: (a) Pruned ?-subgraph, and (b) its re-coloring after refinement which changes the color of $\hat{t}_0 \vdash \neg i$ to T.

Optimization. In some cases, the ?-subgraphs can be pruned further at the end of an iteration, before they are refined, based on the product graph computed in the same iteration and its coloring. One possible reduction is the following. We say that a node n of a ?-subgraph *appears as a sub-node* of some node in the product graph if some node in the product graph shares the same subformula as n and its state (which is a pair of states of the individual components) consists of the state of n. Now, if a node n of a ?-subgraph does *not* appear as a sub-node of any node in the corresponding product graph, then n can be removed from its ?-subgraph as well. For example, at the end of the first iteration of Example 5.11, the nodes $\hat{t}_0 \vdash \Diamond o$ and $\hat{t}_0 \vdash o$ of the ?-subgraph depicted in Fig. 8(a) can be removed since they are not sub-nodes of any node in the product graph of the first iteration, depicted in Fig. 8(b).

Moreover, if a node n of a ?-subgraph only appears as a sub-node of nodes that are colored by T (resp. F) in the corresponding product graph, then n can be colored the same way in its ?-subgraph. For example, at the end of the first iteration of Example 5.11, the node $\hat{t}_1 \vdash o$ of the ?-subgraph depicted in Fig. 8(a) can be colored T since it only appears as a sub-node of $(s_2, \hat{t}_1) \vdash o$, which is colored T, in the product graph of the first iteration, depicted in Fig. 8(b). Furthermore, at the end of the second iteration, the nodes $\hat{t}_1 \vdash \neg i \vee \Diamond o$ and $\hat{t}_1 \vdash \Diamond o$ of the ?-subgraph from the second iteration, depicted in Fig. 8(a), can be colored T since the corresponding nodes in the product graph of the second iteration, depicted in Fig. 8(c), are all colored T. This allows further pruning of the ?-subgraph which removes the entire subgraph whose root is the node $\hat{t}_1 \vdash \neg i \vee \Diamond o$, since once this node is colored T, it is no longer a witnessing son of the initial node.

Applying these optimizations in Example 5.11 allows us to prune the ?-subgraph depicted in Fig. 8(a) at the end of the second iteration, before refinement is applied, into the one depicted in Fig. 9(a). The refinement is then applied to this pruned subgraph. As before, it changes the color of the node $\hat{t}_0 \vdash \neg i$ to T (see Example 5.11). The third iteration starts from the resulting subgraph. Fig. 9(b) exhibits its coloring, computed in the third iteration. The initial node is now colored T. Thus, due to these optimizations, the property is verified without the need to construct the product graph in the third iteration.

Relation to Assume-Guarantee Reasoning. As explained in Section 5.1, an abstraction of a component M_i (which comprises the environment of the other component) can be viewed as providing an assumption on M_i. From this point of view, our compositional abstraction-refinement resembles iterative AG reasoning: When applying abstraction-refinement on one or both of the components, the result is an (asymmetric or symmetric) automatic mechanism for assumption generation. In each iteration, more information about the component is revealed based on the cause for the indefinite result. The use of conservative abstractions guarantees that the assumption describes the component correctly (by construction). Thus unlike typical AG reasoning, this

need not be verified. Moreover, each iteration of our approach benefits from the compositional model checking described in the previous section.

6. Extensions

We now describe two extensions of our compositional algorithm. In both cases we refer to the case where the components are concrete. However, similar extensions are applicable in the abstract case.

6.1. Labeled Transition Systems

In this section we sketch how our compositional model checking technique can be applied to Labeled Transition Systems (LTSs).

A Labeled Transition System (LTS) is a tuple $M = (A, S, s^0, R)$, where A is a set of observable actions, called the *alphabet* of M, S is a finite set of states, $s^0 \in S$ is the initial state, and $R \subseteq S \times (A \cup \{\tau\}) \times S$ is a transition relation. τ denotes a local action, unobservable to M's environment.

Composition of LTSs is defined as follows. Let $M_1 = (A_1, S_1, s_1^0, R_1)$ and $M_2 = (A_2, S_2, s_2^0, R_2)$ be two LTSs. Their composition, denoted $M_1 \| M_2$, is the LTS $M = (A, S, s^0, R)$, where $A = A_1 \cup A_2$, $S = S_1 \times S_2$, $s^0 = (s_1^0, s_2^0)$, and

$$\begin{aligned} R \quad = \quad & \{((s_1, s_2), a, (t_1, t_2)) \mid a \in A_1 \cap A_2 \text{ and } (s_1, a, t_1) \in R_1 \text{ and } (s_2, a, t_2) \in R_2\} \\ & \cup \{((s_1, s_2), a, (t_1, s_2)) \mid a \in (A_1 \setminus A_2) \cup \{\tau\} \text{ and } (s_1, a, t_1) \in R_1 \text{ and } s_2 \in S_2\} \\ & \cup \{((s_1, s_2), a, (s_1, t_2)) \mid a \in (A_2 \setminus A_1) \cup \{\tau\} \text{ and } (s_2, a, t_2) \in R_2 \text{ and } s_1 \in S_1\} \end{aligned}$$

Actions in $A_1 \cap A_2$ are *joint actions*, while actions in $(A_i \setminus A_{\bar{i}}) \cup \{\tau\}$ are *local actions* of M_i. The first type of transitions represent synchronization on the joint actions, whereas the second and third types represent interleaving of local actions, i.e. actions that do not belong to $A_1 \cap A_2$. In the latter type of transitions, one component proceeds along its transition, while the other component does not change its state.

It is convenient to add τ to the alphabet of the LTS, and distinguish between unobservable actions of different LTSs. As such, we redefine A_i to be $A_i \cup \{\tau_i\}$, and replace any occurrence of τ in R_i by τ_i.

Now, an LTS M_i can be viewed as an abstraction $M_i\uparrow$ of $M_1 \| M_2$ by viewing its local transitions (i.e., transitions which are labeled by local actions, including τ_i) as *must* transitions. This is because such transitions do not require synchronization, and thus their existence in the composition does not depend on the other component. The transitions which are labeled by joint actions are viewed as may transitions, as their existence in the composition depends on the other component. We also add to $M_i\uparrow$ additional idle may transitions $(s_i, \tilde{a}, s_i)$ for any $s_i \in S_i$, where $\tilde{a}$ is a new action that represents all the local actions of the other component, $M_{\bar{i}}$, and can therefore synchronize with all of them. The purpose of these idle transitions is to account for the local transitions of $M_{\bar{i}}$, which may or may not exist, and as such, might contribute transitions to the composition. Thus, $M_i\uparrow$ now contains both may and must transitions. The game graphs of the lifted LTSs are constructed and colored as before, and if necessary the ?-subgraphs are computed.

When the product graph of the ?-subgraphs of the lifted LTSs is constructed, every pair of nodes that share the same subformula is composed, with no further restriction on the states of the underlying LTSs. In practice, the nodes of the product graph are computed in a top-down

fashion, hence only reachable pairs are included, and only those that are relevant to the checked property. The edges of the product graph are computed from the edges of the two ?-subgraphs by the same rules as the transitions of $M_1 \| M_2$. As a result, if for some node one ?-subgraph contains an outgoing may edge which is labeled by a joint action or $\tilde{a}$, but the other ?-subgraph has no matching edge, either since a corresponding transition does not exist in the corresponding component, or since it was pruned from the ?-subgraph, then the edge will not be included in the product graph. In addition, outgoing must edges of a node in one ?-subgraph which are labeled by local actions can be pruned as well: for such edges, the game graph of the other component always contains a corresponding (idle may) edge. However, the corresponding edge might be pruned when the ?-subgraph is constructed (if the source node of the edge has a definite color or if its target node is a non-witnessing son), in which case it will also be pruned from the product graph. This allows pruning of both may and must edges in the product graph. The remaining edges are all must edges (as they refer to the full system). Since the product graph is computed top-down, the pruning of the edges also results in pruning of the nodes.

The fact that there is no restriction in terms of the LTS states on which nodes of the ?-subgraphs are composed reflects the different synchronization mechanism of LTSs. However, our approach can still yield reduction in this case as well, due to the fact that each of the ?-subgraphs is pruned. Thus, the resulting product graph is also pruned compared to the full game graph for $M_1 \| M_2$.

6.2. Symbolic Algorithm

In this section we describe how the ideas behind the compositional algorithm can be adapted to symbolic model checking as well, based on sets and set-manipulation. This is useful since sets can be effectively described and manipulated by means of Boolean functions and their BDD representation [33] for example. We consider the *alternation-free* fragment of the μ-calculus, where nesting of fixpoints is not allowed. Namely, for every subformula $\eta Z.\psi$, no variable other than Z occurs freely in ψ. We start with some background on symbolic 3-valued model checking. Note that in this section we go back to modelling systems as Kripke structures (or KMTSs), rather than LTSs.

Symbolic 3-Valued Model Checking. Let M be a KMTS and φ an alternation-free μ-calculus formula. We first introduce set notation for the semantics: We denote by $[\![\varphi]\!]^M_{\mathrm{tt}}$, $[\![\varphi]\!]^M_{\mathrm{ff}}$, and $[\![\varphi]\!]^M_{\perp}$, resp., the sets of states in M for which the truth value of φ is tt, ff, or $\perp$, resp. That is, $[\![\varphi]\!]^M_{\mathrm{tt}} = \{s \mid [\![\varphi]\!]^M_3(s) = \mathrm{tt}\}$, and similarly for ff and $\perp$. Symbolic 3-valued model checking is done by computing these sets for the desired property φ. We use the following notation. For $B \subseteq S$: $ax(B) = \{s \mid \forall s' : sR^-s' \Rightarrow B(s')\}$ and $ex(B) = \{s \mid \exists s' : sR^+s' \wedge B(s')\}$. The sets $[\![\varphi]\!]^M_{\mathrm{tt}}$ and $[\![\varphi]\!]^M_{\mathrm{ff}}$ are computed in a bottom-up fashion (i.e., from the simplest subformulas to the more complex ones) as follows.

$$[\![l]\!]^M_{\mathrm{tt}} = \{s \in S : l \in L(s)\} \qquad [\![l]\!]^M_{\mathrm{ff}} = \{s \in S : \neg l \in L(s)\} \qquad \text{for } l \in Lit$$
$$[\![\varphi_1 \wedge \varphi_2]\!]^M_{\mathrm{tt}} = [\![\varphi_1]\!]^M_{\mathrm{tt}} \cap [\![\varphi_2]\!]^M_{\mathrm{tt}} \qquad [\![\varphi_1 \wedge \varphi_2]\!]^M_{\mathrm{ff}} = [\![\varphi_1]\!]^M_{\mathrm{ff}} \cup [\![\varphi_2]\!]^M_{\mathrm{ff}}$$
$$[\![\varphi_1 \vee \varphi_2]\!]^M_{\mathrm{tt}} = [\![\varphi_1]\!]^M_{\mathrm{tt}} \cup [\![\varphi_2]\!]^M_{\mathrm{tt}} \qquad [\![\varphi_1 \vee \varphi_2]\!]^M_{\mathrm{ff}} = [\![\varphi_1]\!]^M_{\mathrm{ff}} \cap [\![\varphi_2]\!]^M_{\mathrm{ff}}$$
$$[\![\Box\varphi_1]\!]^M_{\mathrm{tt}} = ax([\![\varphi_1]\!]^M_{\mathrm{tt}}) \qquad [\![\Box\varphi_1]\!]^M_{\mathrm{ff}} = ex([\![\varphi_1]\!]^M_{\mathrm{ff}})$$
$$[\![\Diamond\varphi_1]\!]^M_{\mathrm{tt}} = ex([\![\varphi_1]\!]^M_{\mathrm{tt}}) \qquad [\![\Diamond\varphi_1]\!]^M_{\mathrm{ff}} = ax([\![\varphi_1]\!]^M_{\mathrm{ff}})$$

For $v \in \{\mathrm{tt}, \mathrm{ff}\}$ and $\eta \in \{\mu, \nu\}$, the set $[\![\eta Z.\varphi_1]\!]^M_v$ is computed as the fixpoint of the sequence $[\![Z^0]\!]^M_v, [\![Z^1]\!]^M_v, \ldots, [\![Z^i]\!]^M_v \ldots$, where

$$[\![Z^0]\!]^M_{\mathrm{tt}} = \begin{cases} \emptyset, & \text{if } \eta = \mu \\ S, & \text{if } \eta = \nu \end{cases} \qquad [\![Z^0]\!]^M_{\mathrm{ff}} = \begin{cases} S, & \text{if } \eta = \mu \\ \emptyset, & \text{if } \eta = \nu \end{cases}$$

Moreover, for $i \geq 1$: $[\![Z^i]\!]^M_v = [\![\varphi_1[Z \mapsto Z^{i-1}]]\!]^M_v$.

The fixpoints exist since the transformers used in the fixpoint definitions of $[\![\eta Z.\varphi_1]\!]^M_v$ are monotonic and the state space is finite.

Furthermore, for every alternation-free μ-calculus formula φ, $[\![\varphi]\!]^M_\perp$ is computed as $S \setminus ([\![\varphi]\!]^M_{\mathrm{tt}} \cup [\![\varphi]\!]^M_{\mathrm{ff}})$.

The model checking result of a closed formula φ in a KMTS M is then tt if $s^0 \in [\![\varphi]\!]^M_{\mathrm{tt}}$, it is ff if $s^0 \in [\![\varphi]\!]^M_{\mathrm{ff}}$, and it is $\perp$ otherwise (where s^0 is the initial state of M).

Symbolic Compositional Model Checking. We first describe the basic symbolic compositional algorithm without getting into the details of the symbolic representation. The details of a symbolic representation by means of Boolean functions, as well as an optimization of the algorithm are provided later on.

The first step in the symbolic compositional algorithm is the same as in the game-based algorithm. Namely, each (concrete) model M_i is lifted into an abstract model $M_i\uparrow$ and is model checked separately, except that the symbolic 3-valued model checking is used. As before, if the value of φ on either $M_1\uparrow$ or $M_2\uparrow$ is definite (tt or ff) then the same value holds in $M_1 \| M_2$ and the algorithm terminates with an answer.

However, if the value of φ on both $M_1\uparrow$ and $M_2\uparrow$ is indefinite, then its value on $M_1 \| M_2$ is unknown. In this case, the composition of the components needs to be considered. This is performed in the second step, where, similarly to the game-based algorithm, instead of considering the full system, we restrict the re-evaluation of the formula to the "indefinite" parts of $M_1 \| M_2$.

First, for each subformula ψ of φ, we deduce the tt, ff and $\perp$-sets of the composed state space, denoted $TS(\psi)$, $FS(\psi)$ and $US(\psi)$ resp., from the intermediate results of the symbolic model checking of $M_1\uparrow$ and $M_2\uparrow$. The tt-set of $M_1 \| M_2$ consists of the states (s_1, s_2) where either s_1 belongs to the tt-set in $M_1\uparrow$ or s_2 belongs to the tt-set in $M_2\uparrow$. Similarly for the ff-set. The $\perp$-set consists of the rest. More specifically, let S denote the state space of $M_1 \| M_2$. Then

$$TS(\psi) = \{(s_1, s_2) \in S \mid s_1 \in [\![\psi]\!]^{M_1\Uparrow}_{\mathrm{tt}}\} \cup \{(s_1, s_2) \in S \mid s_2 \in [\![\psi]\!]^{M_2\Uparrow}_{\mathrm{tt}}\}$$

$$FS(\psi) = \{(s_1, s_2) \in S \mid s_1 \in [\![\psi]\!]^{M_1\Uparrow}_{\mathrm{ff}}\} \cup \{(s_1, s_2) \in S \mid s_2 \in [\![\psi]\!]^{M_2\Uparrow}_{\mathrm{ff}}\}$$

$$US(\psi) = \{(s_1, s_2) \in S \mid s_1 \in [\![\psi]\!]^{M_1\Uparrow}_{\perp}\} \cap \{(s_1, s_2) \in S \mid s_2 \in [\![\psi]\!]^{M_2\Uparrow}_{\perp}\}$$

This computation resembles Step 2(2) in the game-based algorithm, in which the product graph of the ?-subgraphs is constructed (see Fig. 4).

Similarly to Step 2(3) in the game-based algorithm, where the product graph is colored, we now need to re-evaluate the value of φ in its $\perp$-set in order to determine its value in (s^0_1, s^0_2). This is done by the symbolic model checking algorithm, which re-evaluates all the subformulas in a bottom-up fashion (from simpler subformulas to more complex ones). The re-evaluation is over $M_1 \| M_2$. Yet, no re-evaluation is needed for subformulas whose $\perp$-sets are empty (e.g., literals), or for their subformulas. In addition, for the computation of $[\![\mu Z.\varphi_1]\!]^{M_1|M_2}_{\mathrm{tt}}$ for a least fixpoint formula $\mu Z.\varphi_1$, instead of starting from $[\![Z^0]\!]^{M_1|M_2}_{\mathrm{tt}} = \emptyset$, we start from $[\![Z^0]\!]^{M_1|M_2}_{\mathrm{tt}} = TS(\mu Z.\varphi_1)$, which comprises a lower bound on the result. Similarly, for the computation of $[\![\mu Z.\varphi_1]\!]^{M_1|M_2}_{\mathrm{ff}}$, instead of starting from $[\![Z^0]\!]^{M_1|M_2}_{\mathrm{ff}} = S$, we start from $[\![Z^0]\!]^{M_1|M_2}_{\mathrm{ff}} = FS(\mu Z.\varphi_1) \cup US(\mu Z.\varphi_1)$, which comprises an upper bound on the result. This has a positive effect on the convergence time of the fixpoint computations. A dual adaptation is done when considering greatest fixpoint formulas.

Once the re-evaluation is completed, the algorithm terminates with an answer, depending on whether (s^0_1, s^0_2) belongs to $[\![\varphi]\!]^{M_1|M_2}_{\mathrm{tt}}$ or to $[\![\varphi]\!]^{M_1|M_2}_{\mathrm{ff}}$.

Symbolic Representation by Means of Boolean Functions. In order to exemplify the symbolic algorithm in a more concrete way, we describe how it is applied when using a symbolic representation of models and sets by means of Boolean functions (or BDDs). We assume that the set of states of each model M_i is defined over a set of Boolean variables $\mathcal{V}_i$, some of which are joint variables. That is, $S_i = S(\mathcal{V}_i)$, where $S(\mathcal{V})$ denotes the set of all assignments to $\mathcal{V}$. Each state s_i of M_i is encoded by an assignment to $\mathcal{V}_i$. If the set of states is not given this way, it can simply be encoded this way. Each literal corresponds to a Boolean variable (positive or negated). We assume further that the joint literals of the components, $Lit_1 \cap Lit_2$, correspond exactly to the joint variables $\mathcal{V}_1 \cap \mathcal{V}_2$. The initial state and the transition relation of M_i are also given as Boolean formulas over $\mathcal{V}_i$, $s_i^0(\mathcal{V}_i)$ and $R_i(\mathcal{V}_i, \mathcal{V}_i')$ resp. (where $\mathcal{V}_i'$ is a copy of the variables used to represent the variables in the next-state).

Recall that each state in the state space of $M_1 \| M_2$ is a pair $(s_1, s_2) \in S_1 \times S_2$, that fulfills the composability restriction: $L_1(s_1) \cap Lit_2 = L_2(s_2) \cap Lit_1$. Since the joint literals correspond to the joint variables, the composability requirement of $s_1 \in S_1$ and $s_2 \in S_2$ simply means that s_1 and s_2 agree on the joint variables, $\mathcal{V}_1 \cap \mathcal{V}_2$. In this case, a state (s_1, s_2) of the composed system $M_1 \| M_2$ can simply be viewed as an assignment to the union of the variables, $\mathcal{V} \triangleq \mathcal{V}_1 \cup \mathcal{V}_2$. This immediately implies that there is no disagreement over the joint literals since the joint variables get assigned just once.

Thus, in $M_1 \| M_2$, $S = S(\mathcal{V})$. Moreover, $s^0(\mathcal{V}) = s_1^0(\mathcal{V}_1) \wedge s_2^0(\mathcal{V}_2)$ and $R(\mathcal{V}, \mathcal{V}') = R_1(\mathcal{V}_1, \mathcal{V}_1') \wedge R_2(\mathcal{V}_2, \mathcal{V}_2')$.

In this setting, if the sets $[\![\psi]\!]_{\mathrm{tt}}^{M_i^\uparrow}$, $[\![\psi]\!]_{\mathrm{ff}}^{M_i^\uparrow}$ and $[\![\psi]\!]_{\perp}^{M_i^\uparrow}$ (over the states of M_i) are given as Boolean formulas (or BDD's) over $\mathcal{V}_i$, then the computation of $TS(\psi)$, $FS(\psi)$ and $US(\psi)$ (over the states of $M_1 \| M_2$) for the subformulas ψ of φ in Step 2(2) of the algorithm can be reformulated as follows:

$$TS(\psi)(\mathcal{V}) = [\![\psi]\!]_{\mathrm{tt}}^{M_1^\uparrow}(\mathcal{V}_1) \vee [\![\psi]\!]_{\mathrm{tt}}^{M_2^\uparrow}(\mathcal{V}_2)$$

$$FS(\psi)(\mathcal{V}) = [\![\psi]\!]_{\mathrm{ff}}^{M_1^\uparrow}(\mathcal{V}_1) \vee [\![\psi]\!]_{\mathrm{ff}}^{M_2^\uparrow}(\mathcal{V}_2)$$

$$US(\psi)(\mathcal{V}) = [\![\psi]\!]_{\perp}^{M_1^\uparrow}(\mathcal{V}_1) \wedge [\![\psi]\!]_{\perp}^{M_2^\uparrow}(\mathcal{V}_2)$$

Optimization: Further Pruning of the Re-evaluated Sets. Not all the states that belong to the $\perp$-set of a subformula ψ of φ indeed affect the value of φ in (s_1^0, s_2^0). For example, if $\varphi = \Box\varphi_1$ and $s' \in US(\varphi_1)$, but there is no transition from (s_1^0, s_2^0) to s', then the value of φ_1 in s' is irrelevant. To prune the sets that require re-evaluation in the second step of the algorithm, we therefore first compute for each subformula ψ of φ the *relevant* set $RS(\psi)$ of states. These are the states of $M_1 \| M_2$ such that the value of ψ in them is needed in order to re-evaluate φ in (s_1^0, s_2^0). We then restrict the symbolic model checking algorithm to these sets.

The relevant sets are computed by ComputeRelevant (see Fig. 10). Initially, all relevant sets are set to $\emptyset$. The computation proceeds along the subformulas in a top-down manner (i.e., from more complex subformulas to simpler ones), starting from φ, for which the initial relevant set *newRS* is $\{(s_1^0, s_2^0)\}$. Given a relevant set *newRS* of a formula ψ, *newRS* is first updated to the states that are indeed new, i.e. not already in $RS(\psi)$ (Line 1). If the remaining set is nonempty (Line 2), it is added to $RS(\psi)$ (Line 3). The relevant sets of the subformulas of ψ are then computed as those that are needed for the re-evaluation of ψ in the *indefinite* subset of its relevant set. Note that the relevant set might include states for which the corresponding subformula has definite values. However, the search is pruned in such states. Namely, the states that affect their (definite) values are no longer added to the corresponding relevant sets.

Algorithm ComputeRelevant (ψ : alternation-free formula, $newRS$: relevant set)

1: $newRS$:= $newRS \setminus RS(\psi)$

% keep in $newRS$ only new states, i.e., that are not already in $RS(\psi)$.

2: If $newRS = \emptyset$ then return

3: $RS(\psi)$:= $RS(\psi) \cup newRS$

4: If $\psi = l \in Lit$ then return

5: If $\psi = \Box\psi_1$ or $\psi = \Diamond\psi_1$ then

6: ComputeRelevant(ψ_1, ForwardStep($newRS \cap US(\psi)$, R))

% forward step from the indefinite states in $newRS$.

7: Else if $\psi = \psi_1 \wedge \psi_2$ or $\psi = \psi_1 \vee \psi_2$ then for each $i \in \{1,2\}$ do

8: ComputeRelevant(ψ_i, $newRS \cap US(\psi)$)

9: Else if $\psi = \mu Z.\psi_1$ or $\psi = \nu Z.\psi_1$ then ComputeRelevant(ψ_1, $newRS$)

10: Else if $\psi = Z$ then ComputeRelevant($fp_\varphi(Z)$, $newRS$)

Figure 10: Algorithm for the computation of relevant sets. When applied on $(\varphi, \{(s_1^0, s_2^0)\})$, it computes the relevant sets for the subformulas of φ.

More specifically, the relevant sets of the subformulas of ψ are computed as follows. if ψ is a literal, then the computation terminates (Line 4) as no subformulas exist. If ψ is a $\Box$ or a $\Diamond$ formula, then the relevant set of its subformula ψ_1 is determined by a (symbolic) forward step along the transitions of $M_1 \| M_2$ from the indefinite subset of the relevant set of ψ (Line 6). The forward step is computed by ForwardStep. For $B \subseteq S$ and $R \subseteq S \times S$: ForwardStep(B, R) $= \{s' \mid \exists s : B(s) \wedge sRs'\}$. If ψ is a $\wedge$ or a $\vee$ formula, where the value depends only on the subformulas and not on the model's transitions, then the subformulas of ψ "inherit" the relevant set of ψ, except that it is intersected with the $\bot$-set of ψ, since the search is pruned in the definite states (Line 8). For fixpoint formulas, a (symbolic) iterative computation takes place to determine the relevant sets, where each iteration adds to $RS(\psi)$ the states that affect the indefinite values for the states that were already in $RS(\psi)$ in the previous iteration (Lines 9 and 10). The iterative computation ends when the newly computed set $newRS$ is a subset of $RS(\psi)$, i.e., no additional states were encountered (Line 2).

Remark 6.1. In comparison to the game-based algorithm, the relevant sets of the subformulas of φ provide a symbolic description of the product graph $G_\|$, which describes the "indefinite" subgraph of the game graph over $M_1 \| M_2$.

Now, during the re-evaluation of φ on $M_1 \| M_2$ with the symbolic model checking, for each subformula ψ of φ we consider only its relevant subset $RS(\psi)$. Namely, after each step of the symbolic algorithm, which computes (a subset of) $[\![\psi]\!]_{\mathrm{tt}}^{M_1|M_2}$ and $[\![\psi]\!]_{\mathrm{ff}}^{M_1|M_2}$ for some subformula ψ, these sets are restricted to $RS(\psi)$ (by intersection). The restricted sets are extended by the previously computed definite sets, also restricted to the relevant sets, $TS(\psi) \cap RS(\psi)$ and $FS(\psi) \cap RS(\psi)$ resp. More precisely,

$$[\![\psi]\!]_{\mathrm{tt}}^{M_1|M_2} := ([\![\psi]\!]_{\mathrm{tt}}^{M_1|M_2} \cap RS(\psi)) \cup (TS(\psi) \cap RS(\psi)) = ([\![\psi]\!]_{\mathrm{tt}}^{M_1|M_2} \cup TS(\psi)) \cap RS(\psi)$$
$$[\![\psi]\!]_{\mathrm{ff}}^{M_1|M_2} := ([\![\psi]\!]_{\mathrm{ff}}^{M_1|M_2} \cap RS(\psi)) \cup (FS(\psi) \cap RS(\psi)) = ([\![\psi]\!]_{\mathrm{ff}}^{M_1|M_2} \cup FS(\psi)) \cap RS(\psi)$$

The extension by the previously computed definite sets is needed since the relevant sets of the subformulas of ψ only suffice to determine the value of ψ in the *indefinite* subset of $RS(\psi)$. Thus, the definite subsets of $RS(\psi)$ need to be added separately. This adaptation applies in particular to the computation of fixpoints. Here, after $[\![Z^i]\!]_{\mathrm{tt}}^{M_1|M_2}$ and $[\![Z^i]\!]_{\mathrm{ff}}^{M_1|M_2}$ are computed, they are intersected with $RS(fp_\varphi(Z))$ and extended by $TS(fp_\varphi(Z)) \cap RS(fp_\varphi(Z))$, resp. $FS(fp_\varphi(Z)) \cap RS(fp_\varphi(Z))$.

The symbolic model checking proceeds with the updated sets, which has the advantage of maintaining the sets small and focusing the computational effort on states that indeed influence the value of φ in (s_1^0, s_2^0).

Remark 6.2. When considering the abstract case, the difference in the second step of the algorithm is that during the re-evaluation of φ on $\hat{M}_1 \| \hat{M}_2$, which is now an abstract model, we need to use R^- and R^+ (instead of R). Specifically, when considering the symbolic representation by means of Boolean functions as described above, $R^-(\mathcal{V}, \mathcal{V}') = R_1^-(\mathcal{V}_1, \mathcal{V}_1') \wedge R_2^-(\mathcal{V}_2, \mathcal{V}_2')$, and $R^+(\mathcal{V}, \mathcal{V}') = R_1^+(\mathcal{V}_1, \mathcal{V}_1') \wedge R_2^+(\mathcal{V}_2, \mathcal{V}_2')$.

Moreover, in the abstract case, the result of the re-evaluation might still be indefinite, meaning that $(\hat{s}_1^0, \hat{s}_2^0)$ is in $[\![\varphi]\!]_\perp^{\hat{M}_1|\hat{M}_2}$, in which case refinement is needed. Refinement can again be performed compositionally as in step 3 of the compositional abstraction-refinement algorithm presented in Fig. 7. More specifically, [34] suggests a symbolic 3-valued abstraction-refinement algorithm for CTL. Similarly to the game-based algorithm of [9, 10], they use the symbolic model checking algorithm in order to also return a failure state and a failure cause which guide the refinement, in case the result is indefinite. This can be generalized to the alternation-free μ-calculus in a straightforward manner. Refinement can then be performed as before.

7. Concluding Remarks

This paper suggests a new approach for compositional verification, which utilizes the game-based techniques developed for 3-valued model checking of abstract models. A symbolic variant of the approach is discussed as well.

Our method is described for systems composed of two components, but it can be extended to the composition of n components. The main difference is that in the general case, after combining the ?-subgraphs of a subset of the (concrete) components, we will still be left with an abstract graph containing strict may edges and ?-colored terminal nodes due to information missing from the rest of the components.

Our compositional approach tries to resolve as much of the property as possible on each component separately. It then combines the results obtained on all components in order to resolve dependencies between them. The approach is therefore expected to be mostly beneficial when the locality of the property is high. Namely, when the property contains subproperties that can be resolved on one component. This will be reflected in pruning of the game graphs.

To account for cases where the game graphs that arise in the compositional algorithm are still too big (or infinite), we consider abstraction of the components. We combine the compositional algorithm with abstraction-refinement and propose a compositional abstraction-refinement algorithm for the μ-calculus.

Implementation of our algorithm, as well as its evaluation in practice, is the subject of future work.

Acknowledgement. The first author would like to acknowledge the financial support of an IBM Ph.D. Fellowship.

References

[1] S. Shoham, O. Grumberg, Compositional verification and 3-valued abstractions join forces, in: Proceedings of the 14th International Static Analysis Symposium (SAS), Vol. 4634 of Lecture Notes in Computer Science, Springer, Kongens Lyngby, Denmark, 2007, pp. 69–86.

[2] E. Clarke, O. Grumberg, D. Peled, Model Checking, MIT press, 1999.

[3] C. Jones, Specification and design of (parallel) programs, in: Information Processing 83: Proc. of the IFIP 9th World Congress, North-Holland, 1983, pp. 321–332.

[4] A. Pnueli, In transition from global to modular temporal reasoning about programs, in: K. R. Apt (Ed.), Logics and Models of Concurrent Systems, Vol. 13 of NATO ASI series F, Springer-Verlag, 1984.

[5] J. M. Cobleigh, D. Giannakopoulou, C. S. Pasareanu, Learning assumptions for compositional verification, in: Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03), Vol. 2619 of LNCS, Warsaw, Poland, 2003, pp. 331–346.

[6] R. Alur, P. Madhusudan, W. Nam, Symbolic compositional verification by learning assumptions, in: Computer Aided verification (CAV'05), Vol. 3576 of LNCS, 2005, pp. 548–562.

[7] S. Chaki, E. M. Clarke, N. Sinha, P. Thati, Automated assume-guarantee reasoning for simulation conformance, in: Computer Aided verification (CAV'05), Vol. 3576 of LNCS, 2005, pp. 534–547.

[8] S. Shoham, O. Grumberg, A game-based framework for CTL counterexamples and 3-valued abstraction-refinement, ACM Transactions on Computational Logic (TOCL) 9 (1) (2007) 1.

[9] O. Grumberg, M. Lange, M. Leucker, S. Shoham, Don't know in the μ-calculus, in: 6th international conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05), Vol. 3385 of LNCS, Paris, France, 2005, pp. 233–249.

[10] O. Grumberg, M. Lange, M. Leucker, S. Shoham, When not losing is better than winning: Abstraction and refinement for the full μ-calculus, Information and Compuatation 205 (2007) 1130–1148.

[11] O. Grumberg, D. Long, Model checking and modular verification, ACM Trans. on Programming Languages and Systems 16 (3) (1994) 843–871.

[12] H. Barringer, D. Giannakopoulou, C. Pasareanu, Proof rules for automated compositional verification through learning, in: 2nd Workshop on Specification and Verification of Component-Based Systems (SAVCBS), 2003.

[13] D. Giannakopoulou, C. S. Pasareanu, H. Barringer, Assumption generation for software component verification, in: ASE, Vol. 00, 2002, p. 3.

[14] R. Alur, P. Cerny, P. Madhusudan, W. Nam, Synthesis of interface specifications for Java classes, in: POPL '05: Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM Press, New York, NY, USA, 2005, pp. 98–109.

[15] A. Farzan, Y. Chen, E. M. Clarke, Y. Tsan, B. 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), Vol. 4963 of LNCS, Springer, Budapest, 2008, pp. 2–17.

[16] M. Gheorghiu Bobaru, C. S. Păsăreanu, D. Giannakopoulou, Automated assume-guarantee reasoning by abstraction refinement, in: CAV '08: Proceedings of the 20th international conference on Computer Aided Verification, Springer-Verlag, 2008, pp. 135–148.

[17] A. Aziz, T. R. Shiple, V. Singhal, A. L. Sangiovanni-vincentelli, Formula-dependent equivalence for compositional CTL model checking, in: David L. Dill (Ed.), Sixth International Conference on Computer-Aided Verification CAV, Vol. 818, Springer-Verlag, Standford, California, USA, 1994, pp. 324–337.

[18] R. Alur, L. de Alfaro, T. A. Henzinger, F. Y. C. Mang, Automating modular verification, in: CONCUR, 1999, pp. 82–97.

[19] L. de Alfaro, T. A. Henzinger, F. Y. C. Mang, Detecting errors before reaching them, in: Computer Aided Verification, 2000, pp. 186–201.

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

[21] R. Alur, T. Henzinger, O. Kupferman, Alternating-time temporal logic, in: Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, Florida, 1997.

[22] L. de Alfaro, P. Godefroid, R. Jagadeesan, Three-valued abstractions of games: Uncertainty, but with precision, in: Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS), 2004, pp. 170–179.

[23] T. Ball, O. Kupferman, An abstraction-refinement framework for multi-agent systems, in: Proc. 21st IEEE Symp. on Logic in Computer Science, 2006.

[24] S. Chaki, E. Clarke, O. Grumberg, J. Ouaknine, N. Sharygina, T. Touili, H. Veith, State/event software verification for branching-time specifications, in: Proceedings of the 5th International Conference on Integrated Formal Methods (IFM), Vol. 3771 of LNCS, 2005, pp. 53–69.

[25] D. Kozen, Results on the propositional μ-calculus, TCS 27.

[26] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, Pacific J. Math 5 (1955) 285–309.

[27] M. Huth, R. Jagadeesan, D. Schmidt, Modal transition systems: A foundation for three-valued program analysis, in: European Symposium on Programming (ESOP'01), Vol. 2028, 2001, pp. 155–169.
[28] P. Godefroid, R. Jagadeesan, Automatic abstraction using generalized model checking, in: Proc. of Conference on Computer-Aided Verification (CAV), Vol. 2404 of LNCS, Springer-Verlag, Copenhagen, Denmark, 2002, pp. 137–150.
[29] K. G. Larsen, B. Thomsen, A modal process logic, in: Proceedings of the Third Annual IEEE Symp. on Logic in Computer Science, LICS 1988, 1988, pp. 203–210.
[30] G. Bruns, P. Godefroid, Model checking partial state spaces with 3-valued temporal logics, in: Computer Aided Verification, 1999, pp. 274–287.
[31] D. Dams, R. Gerth, O. Grumberg, Abstract interpretation of reactive systems, ACM Transactions on Programming Languages and Systems (TOPLAS) 19 (2).
[32] S. Shoham, O. Grumberg, 3-valued abstraction: More precision at less cost, in: Twenty-First Annual IEEE Symposium on Logic In Computer Science (LICS), Seattle, Washington, 2006, pp. 399–410.
[33] R. E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE transactions on Computers C-35 (8) (1986) 677–691.
[34] S. Shoham, O. Grumberg, Monotonic abstraction-refinement for CTL, in: 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Vol. 2988 of LNCS, Barcelona, Spain, 2004, pp. 546–560.