# Compositional Verification and 3-Valued Abstractions Join Forces[⋆]

Sharon Shoham and Orna Grumberg

Computer Science Department, Technion, Haifa, Israel
{sharonsh,orna}@cs.technion.ac.il

**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 $\mu$-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 $\mu$-calculus formula $\varphi$ has a definite value (true or false) on $M_i\!\uparrow$, the same value holds also for $M$. Thus, $\varphi$ 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 $\varphi$. It ignores the parts which can be handled separately. The resulting model is often significantly smaller than the full system.

We explain how our compositional approach can be combined with abstraction, 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 [11] is a useful approach for verifying properties of systems. 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 $\mu$-calculus with respect to a given system.

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 [22,25] 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 [12,5,10] (see the related work section for more details). In contrast, our approach is

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

based on techniques taken from the 3-valued game-based model checking for abstract models [26,18,19].

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). 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 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 [20]. 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.

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 $\mu$-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 $\mu$-calculus formula $\varphi$ has a definite value (tt or ff) on $M_i\!\uparrow$, the same value holds also for $M$. Thus, $\varphi$ 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 $\varphi$ 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.

More specifically, our approach is based on the 3-valued game for model checking of $\mu$-calculus, suggested in [18,19]. 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 $\psi$ is a subformula of the checked formula, s.t. the value of $\psi$ 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 $\psi$ 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) $\psi$. 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.

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 $\perp$, 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 [26,18,19] in finding the *cause* for the indefinite result on the composed model. However, the refinement is applied on each of the components separately. Moreover, we adopt the incremental approach of [26,18,19] 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.

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 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* $\mu$-calculus.

**Related Work.** Recently, [12] followed by [5,10], 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 [7], where the AG rule used is symmetric. Assumption generation in a more general setting is addressed in [16,2]. These works are all restricted to *universal safety* properties. The learning algorithms used in these works also perform some kind of an abstraction-refinement. However, these algorithms are not specifically tailored for verification. In particular, 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. Moreover, we increase the modularity of the model checking step by using the game-based approach, which also enables an incremental analysis. Most importantly, our approach is applicable to the *full* $\mu$-calculus.

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 [1] the authors merge component's states that share the same value for a given CTL formula in all environments, thus minimizing the component. In [3] 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.

[15] 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.

[24] 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.

In [4] the authors suggest to use game structures to reason about composition of components. [14,6] 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.

[9] 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.

## 2 Preliminaries

**$\mu$-calculus.** [23] 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 *$\mu$-calculus* in *negation normal form* over $AP$ is defined by:

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

where $l \in Lit$ and $Z \in \mathcal{V}$. Intuitively, $\Box$ stands for "all successors", and $\Diamond$ stands for "exists a successor". $\mu$ denotes a least fixpoint, whereas $\nu$ denotes greatest fixpoint. We will also write $\eta$ for either $\mu$ or $\nu$. Let $\mathcal{L}_\mu$ denote the set of *closed* formulas generated by the above grammar, where the fixpoint quantifiers $\mu$ and $\nu$ 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 $fp(Z) = \eta Z.\psi$ of $\varphi$, where the set $Sub(\varphi)$ of *subformulas* of $\varphi$ is defined in the usual way.

**Concrete Semantics.** Concrete systems are typically modelled as *Kripke structures*. A Kripke structure [11] 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 $\{\text{tt}, \text{ff}\}$. $[\![\varphi]\!]^M(s) = \text{tt}\ (= \text{ff})$ means that the formula $\varphi$ is true (false) in the state $s$ of the Kripke structure $M$. If $[\![\varphi]\!]^M(s^0) = \text{tt}\ (= \text{ff})$, we say that $M$ satisfies (falsifies) $\varphi$, denoted $M \models \varphi$ ($M \not\models \varphi$).

**3-Valued Abstraction.** In the context of abstraction, *Kripke Modal Transition Systems* [21,17] are often used as abstract models that preserve the $\mu$-calculus.

**Definition 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* $[\![\varphi]\!]_3^M$ of a closed formula $\varphi \in \mathcal{L}_\mu$ w.r.t. a KMTS $M$ is a mapping from $S$ to $\{\text{tt}, \text{ff}, \bot\}$ [8,21]. It preserves both satisfaction (tt) and refutation (ff) from the abstract KMTS to the concrete model it represents. $\bot$ is a new truth value whose meaning is that the truth value over the concrete model is unknown and can be either tt or ff. The interesting cases in the definition of the 3-valued semantics are those of the literals and the modalities.

$$[\![l]\!]_3^M(s) = \text{ tt if } l \in L(s), \text{ ff if } \neg l \in L(s), \perp \text{ otherwise.}$$

$$[\![\Box\psi]\!]_3^M(s) = \begin{cases} \text{tt}, \text{ if } \forall t \in S, \text{ if } sR^-t \text{ then } [\![\psi]\!]_3^M(t) = \text{tt} \\ \text{ff}, \text{ if } \exists t \in S \text{ s.t. } sR^+t \text{ and } [\![\psi]\!]_3^M(t) = \text{ff} \\ \perp, \text{ otherwise} \end{cases}$$

and dually for $\Diamond\psi$ when exchanging tt and ff. The notations $M \models \varphi$ and $M \not\models \varphi$ are used for KMTSs as well. In addition, if $[\![\varphi]\!]_3^M(s^0) = \perp$, the value of $\varphi$ in $M$ is indefinite.

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

**Definition 2 (Mixed Simulation).** *[13,17] Let* $M_1 = (AP, S_1, s_1^0, R_1^+, R_1^-, L_1)$ *and* $M_2 = (AP, S_2, s_2^0, R_2^+, R_2^-, L_2)$ *be two KMTSs, both defined over* $AP$. $H \subseteq S_1 \times S_2$ *is a* mixed simulation *from* $M_1$ *to* $M_2$ *if* $(s_1, s_2) \in H$ *implies:*

1. $L_2(s_2) \subseteq L_1(s_1)$.
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.* $(s_1^0, s_2^0) \in H$, *then* $M_2$ abstracts $M_1$, *denoted* $M_1 \preceq M_2$.

In particular, Def. 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 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 1.** *[17] 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* $[\![\varphi]\!]_3^{M_2}(s_2) \neq \perp \Rightarrow [\![\varphi]\!]_3^{M_1}(s_1) = [\![\varphi]\!]_3^{M_2}(s_2)$.

**Abstract Model Checking.** A 3-valued game-based model checking for the $\mu$-calculus over KMTSs was suggested in [18,19]. 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 satisfaction game, which is a special case of a 3-valued parity game. We omit the details of the 3-valued satisfaction game, but continue with the *game graph*, which presents all the information "relevant" for the model checking.

**Game Graph.** Let $M = (AP, S, s^0, R^+, R^-, L)$ be a KMTS and $\varphi \in \mathcal{L}_\mu^0$. The *game graph* $G_{M \times \varphi}$, or in short $G$, is a graph $(N, n^0, E^+, E^-)$ where $N \subseteq S \times \mathit{Sub}(\varphi)$ is a set of nodes and $E^+ \subseteq E^- \subseteq N \times N$ are sets of must and may edges defined as follows. $n^0 = s^0 \vdash \varphi \in N$ is the initial node. 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 $(s, t) \in R^- \setminus R^+$. Intuitively, the outgoing edges of $s \vdash \psi \in N$ define "subgoals" for checking $\psi$ in $s$.

$$\begin{array}{lll}
\dfrac{s \vdash \psi_0 \vee \psi_1}{s \vdash \psi_i} : i \in \{0,1\} & \dfrac{s \vdash \Diamond\psi}{t \vdash \psi} : \ sR^+t \text{ or } sR^-t & \dfrac{s \vdash \eta Z.\psi}{s \vdash Z} \\
\dfrac{s \vdash \psi_0 \wedge \psi_1}{s \vdash \psi_i} : i \in \{0,1\} & \dfrac{s \vdash \Box\psi}{t \vdash \psi} : \ sR^+t \text{ or } sR^-t & \dfrac{s \vdash Z}{s \vdash \psi} : \text{if } fp(Z) = \eta Z.\psi
\end{array}$$

**Fig. 1.** Rules for game graph construction

If $E^-(n,n')$ ($E^+(n,n')$) then $n'$ is a may (must) son of $n$. A node $n = s \vdash \psi$ in $G_{M\times\varphi}$ is classified as a $\wedge$, $\vee$, $\Box$, $\Diamond$ node, based on $\psi$. If $\psi$ is of the form $Z$ or $\eta Z.\psi'$, $n$ is *deterministic* – it has exactly one son. If $n$ has no outgoing edges, then it is a *terminal node*. In a full game graph this means that either $\psi$ is a literal, or $\psi$ is of the form $\Diamond\psi'$ or $\Box\psi'$, and $s$ has no outgoing transition in $M$.

Fig. 2(b) presents examples of game graphs for $\varphi = \Box(\neg i \vee \Diamond o)$ and the models from Fig. 2(a), where all transitions are considered may transitions.

**Coloring Algorithm.** The model checking algorithms of [18,19] can be viewed as coloring algorithms that label (color) each node $n = s \vdash \psi$ in the game graph by $T$, $F$, ? depending on the truth value of $\psi$ 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 cases the coloring is performed by solving the 3-valued parity game for satisfaction, where each color stands for a possible result in the game. The algorithm of [18] is a generalization of Zielonka's algorithm for solving (2-valued) parity games. In [19], the 3-valued satisfaction game is reduced into two (2-valued) parity games, improving the coloring's complexity. The following formalizes the correctness of the coloring. For a (possibly not closed) formula $\psi$, $\psi^*$ denotes the result of replacing every free occurrence of $Z \in \mathcal{V}$ in $\psi$ by $fp(Z)$. Note that if $\psi$ is closed, then $\psi^* = \psi$.

**Definition 3.** *Let $G_{M\times\varphi}$ be a game graph 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. $[\![\psi^*]\!]_3^M(s) = \perp$ *iff* $\chi(s \vdash \psi) = ?$.

**Theorem 2.** *[18,19] Let $\chi_F$ be the (total) coloring function returned by the coloring algorithm of [18] or [19] for $G_{M\times\varphi}$. Then $\chi_F$ is correct.*

Moreover, in both cases, the final 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. The result of the coloring is demonstrated in Fig. 2(b).

**Refinement.** If the model checking result of an abstract model is indefinite ($\perp$), a refinement is needed. When using the coloring algorithms of [18,19], 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 $\perp$, 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 (see [18,19]).

## 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.

Due to their nature, as algorithms for solving a 3-valued parity game, the coloring algorithms of [18,19] 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. To formalize this, we need the following definitions.

**Definition 4.** *Let $G$ be a game graph and $\chi_F$ its final 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 $\chi_F$.*
**$\vee, \Diamond$:** *the witnessing sons are those colored $T$ or $?$ by $\chi_F$.*
**deterministic:** *the witnessing son is the only son.*

The sons are witnessing in the sense that they suffice to determine the color of the node, thus removing the rest of the node's sons from the graph does not damage the result of the coloring. For example, if a $\wedge$-node or a $\Box$-node has no witnessing sons, meaning all its sons are colored $T$, then we know it should be colored $T$, and this is indeed how the coloring algorithms will color the node when keeping only the witnessing sons. Otherwise, the witnessing sons determine whether the node should be colored $F$ or $?$, thus one can correctly color the node by considering only them.

**Definition 5.** *A subgraph $G'$ of a game graph $G$ is* closed *if every node in $G'$ is either a terminal node, or* all *its witnessing sons (and corresp. edges) from $G$ are also in $G'$.*

**Theorem 3.** *Consider a closed subgraph $G'$ of a game graph $G$ with a partial coloring function $\chi$ which is correct and defined over (at least) all the terminal nodes in $G'$. Then applying the coloring algorithm of [18] or [19] on $G'$ with $\chi$ as an initial coloring results in a correct coloring of $G'$.*

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.

Another notion that we will need later is the following.

**Definition 6 (?-Subgraph).** *Let $G$ be a colored graph whose initial node is colored $?$. The* $?$-subgraph *is the least subgraph $G_?$ 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 $\chi_I$ which is defined over the terminal nodes in $G_?$, and colors them as the coloring function $\chi_F$ of $G$.*

The $?$-subgraph $G_?$ and its initial coloring meet the conditions of Thm. 3. Intuitively, this means that $G_?$ contains *all* the information regarding the indefinite result. Fig. 2(b) provides examples of $?$-subgraphs.

## 4 Compositional Model Checking

In compositional model checking the goal is to verify a formula $\varphi$ 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 7.** *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 8.** *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 of 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 9.** *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).

**Theorem 4.** *$M_1 \| M_2 \preceq M_i\!\uparrow$. The mixed simulation is $\{((s_1, s_2), s_i) \mid (s_1, s_2) \in S\}$.*

Since each $M_i\!\uparrow$ abstracts $M_1 \| M_2$, we are able to first consider each component separately: Thm. 1 ensures that if $\varphi$ 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 $\varphi$ on both $M_1\!\uparrow$ and $M_2\!\uparrow$ is indefinite. This reflects the fact that $\varphi$ 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 [18] or [19], 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. 6) 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 10 (Product Graph).** *Let $G_{?_1}$ and $G_{?_2}$ be two $?$-subgraphs as above with initial nodes $s_1^0 \vdash \varphi$ and $s_2^0 \vdash \varphi$ resp. We define their product to be the least graph $G_\| = (N_\|, n_\|^0, E_\|^+, E_\|^-)$ such that:*

- *$n^0_{\|} = (s^0_1, s^0_2) \vdash \varphi$ 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 $s^0_1$ and $s^0_2$ 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 1.** *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.*

**Definition 11.** *We define the initial coloring function $\chi_I$ of $G_{\|}$ as follows. Let $n$ be a terminal node in $N_{\|}$. If it fulfills case (a) of Prop. 1, 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. $\chi_I$ 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, then the definite color takes over.

Note 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 algorithm to the product graph, all the nodes are colored by definite colors (in fact, a 2-valued coloring can be applied).

**Theorem 5.** *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.*

By Thm. 3, this means that coloring $G_{\|}$ results in a correct result w.r.t. the model checking of $\varphi$ in $M_1 \| M_2$. Thus, to model check $\varphi$ on $M_1 \| M_2$ it remains to color $G_{\|}$. Note that the full graph for $M_1 \| M_2$ is not constructed. To sum up, the algorithm is as follows.

```
Step 1  Model check each $M_i\uparrow$ separately (for $i \in \{1,2\}$):
    1. Construct the game graph $G_i$ for $\varphi$ and $M_i\uparrow$.
    2. Apply the 3-valued coloring on $G_i$. Let $\chi_i$ be the resulting coloring function.
  If $\chi_1(n_1^0)$ or $\chi_2(n_2^0)$ is definite, return the corresp.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 $\chi_{\|}(n_{\|}^0)$.
```

*Example 1.* Consider the components depicted in Fig. 2(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. 2(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. 2(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.

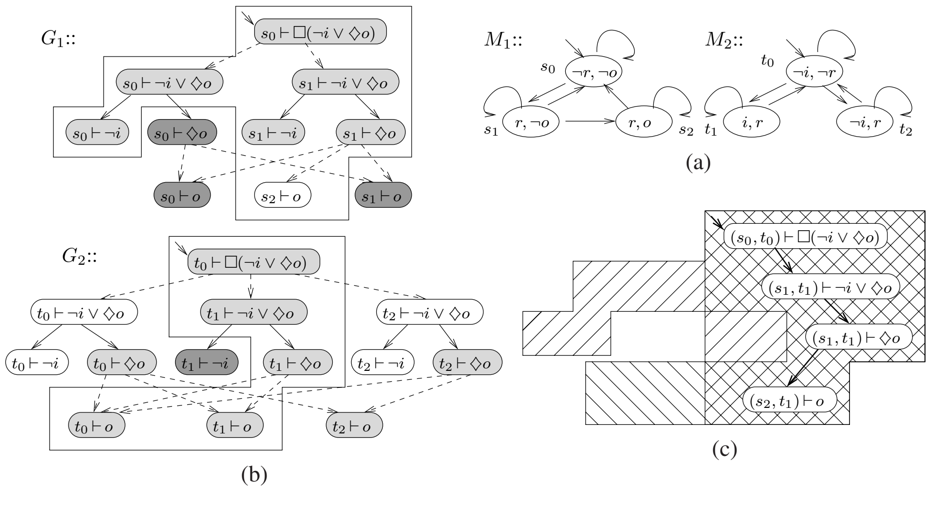

**Fig. 2.** (a) Components, (b) their game graphs and their ?-subgraphs (enclosed by a line), and (c) the product graph. Dashed edges denote may edges which are not must edges. The colors reflect the coloring function: white stands for $T$, dark gray stands for $F$ and light gray stands for ?.

## 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

Composing the ?-subgraphs of two components, as suggested in Section 4, corresponds to refining *all* possible 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. 2, 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. Thus we do not want to resort to that (in this example it is indeed necessary, but in the general case not all the causes for failure need to be eliminated). We now sketch the idea that allows us to only consider the information from $M_2$ that is needed for eliminating the 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 most coarse abstraction of $M_2$ w.r.t. $AP_1 \cap AP_2$, where each state is abstracted by its labeling, restricted to $AP_1 \cap AP_2$.

**Definition 12.** *Let $M_i = (AP_i, S_i, s_i^0, R_i, L_i)$ be a Kripke structure. The* most coarse 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 6.** *$M_i \preceq \hat{M}_i^*$. The mixed simulation is $\{(s_i, L_i(s_i) \cap AP') \mid s_i \in S_i\}$.*

The construction of the most coarse abstraction requires almost no knowledge of the component. More precise transitions can be computed as in [26]. Starting from the most coarse 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 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 most coarse 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. 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. 8. 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 13.** *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 most coarse abstraction w.r.t. $AP_1 \cap AP_2$ (see Def. 12) 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 7.** *Let $\hat{M}_i$ be an appropriate abstract model for $M_i$ w.r.t. $AP_1 \cap AP_2$. Then $M_1 \| M_2 \preceq \hat{M}_1 \| \hat{M}_2$.*

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 14.** *Let $\hat{M}_i = (AP_i, \hat{S}_i, \hat{s}^0_i, R^+_i, R^-_i, \hat{L}_i)$ be a KMTS. We lift $\hat{M}_i$ into a KMTS $\hat{M}_i \!\uparrow = (AP, \hat{S}_i, \hat{s}^0_i, R^+_i \!\uparrow, R^-_i \!\uparrow, \hat{L}_i \!\uparrow)$ 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 8.** $\hat{M}_1 \| \hat{M}_2 \preceq \hat{M}_i\!\uparrow$.

**Corollary 1.** *If $\hat{M}_i$ is an appropriate abstract model for $M_i$ w.r.t. $AP_1 \cap AP_2$, then $M_1 \| M_2 \preceq \hat{M}_i\!\uparrow$.*

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. 1 which characterizes the terminal nodes in the product graph $G_\|$. It is taken into account when determining the initial coloring of $G_\|$.

**Definition 15 (Abstract Product Graph).** *Let $G_{?1}$ and $G_{?2}$ be two abstract $?$-subgraphs as above. Their product graph $G_\| = (N_\|, n^0_\|, E^+_\|, E^-_\|)$ is defined as before, 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, with the addition that a terminal node that fulfills case (c) in the adapted version of Prop. 1 is colored $?$.*

**Theorem 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.*

Along with Thm. 3, this implies that $G_{||}$ can be colored correctly (w.r.t. the model checking of $\varphi$ 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. 7 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 [18,19] 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 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. 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 [26,18,19]. Moreover, in each component we adopt the incremental approach of [26,18,19] 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 following compositional abstraction-refinement loop.

```
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.12).
Step 1  Model check each $\hat{M}_i\uparrow$ separately (for $i \in \{1,2\}$):
          1. Construct the game graph $G_i$ for $\varphi$ and $\hat{M}_i\uparrow$.
          2. Apply the 3-valued coloring on $G_i$. Let $\chi_i$ be the resulting coloring function.
        If $\chi_1(n_1^0)$ or $\chi_2(n_2^0)$ is definite, return the corresp.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 corresp.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.
```

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 can also be used, e.g. with the algorithm of [27].

Using the compositional abstraction-refinement starting from the most coarse 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.

**Theorem 10.** *For finite concrete components, iterating the compositional abstraction-refinement process is guaranteed to terminate with a definite answer.*

## References

1. Aziz, V.A., Shiple, T.R., Sangiovanni-vincentelli, A.L.: Formula-dependent equivalence for compositional CTL model checking. In: Dill, D.L. (ed.) CAV 1994. LNCS, vol. 818, Springer, Heidelberg (1994)
2. Alur, R., Cerny, P., Madhusudan, P., Nam, W.: Synthesis of interface specifications for Java classes. In: POPL (2005)
3. Alur, R., de Alfaro, L., Henzinger, T.A., Mang, F.Y.C.: Automating modular verification. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, Springer, Heidelberg (1999)
4. Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. In: FOCS (1997)
5. Alur, R., Madhusudan, P., Nam, W.: Symbolic compositional verification by learning assumptions. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, Springer, Heidelberg (2005)
6. Ball, T., Kupferman, O.: An abstraction-refinement framework for multi-agent systems. In: LICS (2006)
7. Barringer, H., Giannakopoulou, D., Pasareanu, C.: Proof rules for automated compositional verification through learning. In: SAVCBS (2003)
8. Bruns, G., Godefroid, P.: Model checking partial state spaces with 3-valued temporal logics. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, Springer, Heidelberg (1999)
9. Chaki, S., Clarke, E., Grumberg, O., Ouaknine, J., Sharygina, N., Touili, T., Veith, H.: State/event software verification for branching-time specifications. In: Romijn, J.M.T., Smith, G.P., van de Pol, J. (eds.) IFM 2005. LNCS, vol. 3771, Springer, Heidelberg (2005)
10. Chaki, S., Clarke, E.M., Sinha, N., Thati, P.: Automated assume-guarantee reasoning for simulation conformance. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, Springer, Heidelberg (2005)
11. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press, Cambridge (1999)
12. Cobleigh, J.M., Giannakopoulou, D., Pasareanu, C.S.: Learning assumptions for compositional verification. In: Garavel, H., Hatcliff, J. (eds.) ETAPS 2003 and TACAS 2003. LNCS, vol. 2619, Springer, Heidelberg (2003)
13. Dams, D., Gerth, R., Grumberg, O.: Abstract interpretation of reactive systems. ACM Transactions on Programming Languages and Systems (TOPLAS), 19(2) (1997)
14. de Alfaro, L., Godefroid, P., Jagadeesan, R.: Three-valued abstractions of games: Uncertainty, but with precision. In: LICS (2004)
15. de Alfaro, L., Henzinger, T.A., Mang, F.Y.C.: Detecting errors before reaching them. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, Springer, Heidelberg (2000)

16. Giannakopoulou, D., Pasareanu, C.S., Barringer, H.: Assumption generation for software component verification. In: ASE (2002)
17. Godefroid, P., Jagadeesan, R.: Automatic abstraction using generalized model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, Springer, Heidelberg (2002)
18. Grumberg, O., Lange, M., Leucker, M., Shoham, S.: Don't know in the $\mu$-calculus. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, Springer, Heidelberg (2005)
19. Grumberg, O., Lange, M., Leucker, M., Shoham, S.: When not losing is better than winning: Abstraction and refinement for the full $\mu$-calculus. Information and Compuatation (2007) doi: 10.1016/j.ic.2006.10.009
20. Grumberg, O., Long, D.: Model checking and modular verification. TOPLAS, 16(3) (1994)
21. Huth, M., Jagadeesan, R., Schmidt, D.: Modal transition systems: A foundation for three-valued program analysis. In: Sands, D. (ed.) ESOP 2001 and ETAPS 2001. LNCS, vol. 2028, Springer, Heidelberg (2001)
22. Jones, C.: Specification and design of (parallel) programs. In: IFIP (1983)
23. Kozen, D.: Results on the propositional $\mu$-calculus. TCS, 27 (1983)
24. Li, H.C., Krishnamurthi, S., Fisler, K.: Modular verification of open features using three-valued model checking. Autom. Softw. Eng., 12(3) (2005)
25. Pnueli, A.: In transition for global to modular temporal reasoning about programs. In: Logics and Models of Concurrent Systems, vol. 13 (1984)
26. Shoham, S., Grumberg, O.: A game-based framework for CTL counterexamples and 3-valued abstraction-refinement. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, Springer, Heidelberg (2003) (to appear in TOCL)
27. Shoham, S., Grumberg, O.: 3-valued abstraction: More precision at less cost. In: LICS (2006)