Generalized abstraction-refinement for game-based CTL lifted model checking

Aleksandar S. Dimovski[a], Axel Legay[b], Andrzej Wasowski[c]

[a] *Mother Teresa University, 12 Udarna Brigada 2a, 1000 Skopje, Mkd*
[b] *UCLouvain, Belgium and IRISA/NRIA Rennes, France*
[c] *IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen, Denmark*

Abstract

System families (Software Product Lines) are becoming omnipresent in application areas ranging from embedded system domains to system-level software and communication protocols. Software Product Line methods and architectures allow effective building many custom variants of a software system in these domains. In many of the applications, their rigorous verification and quality assurance are of paramount importance. Lifted model checking for system families is capable of verifying all their variants simultaneously in a single run by exploiting the similarities between the variants. The computational cost of lifted model checking still greatly depends on the number of variants (the size of configuration space), which is often huge. *Variability abstractions* have successfully addressed this configuration space explosion problem, giving rise to smaller abstract variability models with fewer abstract configurations. Abstract variability models are given as modal transition systems, which contain may (over-approximating) and must (under-approximating) transitions. Thus, they preserve both universal and existential CTL properties.

In this work, we bring two main contributions. First, we define a novel game-based approach for variability-specific abstraction and refinement for lifted model checking of the full CTL, interpreted over 3-valued semantics. We propose a direct algorithm for solving a 3-valued (abstract) lifted model checking game. In case the result of model checking an abstract variability model is indefinite, we suggest a new notion of refinement, which eliminates indefinite results. This provides an iterative incremental variability-specific abstraction and refinement framework, where refinement is applied only where indefinite results exist and definite results from previous iterations are reused. Second, we propose a new generalized definition of abstract

variability models, given as so-called generalized modal transition systems, by introducing the notion of (must) hyper-transitions. This results in more precise abstract models in which more CTL formulae can be proved or disproved. We integrate the newly defined generalized abstract variability models in the existing abstraction-refinement framework for game-based lifted model checking of CTL. Finally, we evaluate the practicality of this approach on several system families.

Keywords: Lifted Model Checking, Game-based Model Checking, Variability Abstractions, Automatic Abstraction Refinement

1. Introduction

The strong trend for customization in modern economy leads to construction of many system families. Software Product Line Engineering (SPLE) [1, 2] represents an efficient method to achieve customization of systems by designing families of related systems, known as *variants* or *family members*, from a common code base. Each variant is specified in terms of features (static configuration options) selected for that particular variant. The reuse of code common to multiple variants is maximized. SPLs are particularly popular among the embedded and critical systems (e.g. cars, phones, avionics) [3].

Lifted model checking is an efficient approach for verifying temporal properties of system families. Variability models of system families (SPLs) are given as featured transition systems (FTSs) [4, 5], which represent a widely accepted formalism for specifying the behaviour of all variants of an SPL in a single compact model. Each behaviour in an FTS is associated with the set of variants able to produce it. Given an FTS and a property, a specialized lifted model checking algorithm verifies all variants simultaneously in a single run, and returns precise conclusive results for all individual variants. However, the computational cost of lifted model checking still depends on the number of variants (configurations), which is exponential in the number of features. This is referred as *configuration space explosion problem*.

One of the most successful approaches to fighting the configuration space explosion are so-called *variability abstractions* [6, 7, 8, 9, 10]. They hide some of the configuration details, so that many of the concrete configurations become indistinguishable and can be collapsed into a single abstract configuration (variant). This results in smaller abstract variability models with fewer abstract configurations. However, the over-approximating variability

abstractions introduced in [6, 7, 8] support only the verification of universal LTL properties. CTL [11] is an important branching temporal logic that allows the expression of both universal and existential properties. More importantly, it is a logic for which efficient model checking algorithms exist. Therefore, it is important to devise a technique for efficient lifted model checking of CTL properties. In order to be conservative with respect to the full CTL temporal logic, the abstractions of variability models are given as modal transition systems (MTSs) [9, 10]. They have two types of transitions: *may-transitions* which represent possible transitions in the concrete (FTS) variability model (they occur in some variants), and *must-transitions* which represent the definite transitions in the concrete (FTS) variability model (they occur in all variants). May- and must-transitions correspond to over- and under-approximations, and are needed in order to preserve universal and existential CTL properties, respectively. We consider here the 3-valued semantics for interpreting CTL formulae over abstract variability models (MTSs). This semantics evaluates a formula on an abstract model to either *true*, *false*, or *indefinite*. Abstract variability models are designed to be conservative for both *true* and *false*. However, the *indefinite* answer gives no information on the value of the formula on the concrete model. In this case, a refinement is needed in order to make the abstract models more precise.

In this work, we propose the first variability-specific abstraction-refinement procedure for automatically verifying arbitrary formulae of CTL. To achieve this aim, model checking *games* [12, 13, 14] represent the most suitable framework for defining the refinement. In this way, we establish a brand new connection between games and lifted (SPL) model checking. The refinement is defined by finding the reason for the indefinite result of an algorithm that solves the corresponding model checking game, which is played by two players: Player $\forall$ and Player $\exists$. The goal of Player $\forall$ is either to refute the formula Φ on an abstract model $\mathcal{M}$ or to prevent Player $\exists$ from verifying it. Similarly, the goal of Player $\exists$ is either to verify Φ on $\mathcal{M}$ or to prevent Player $\forall$ from refuting it. The game is played on a *game board*, which consists of configurations of the form (s, Φ') where s is a state of the abstract model $\mathcal{M}$ and Φ' is a subformula of Φ, such that the value of Φ' in s is relevant for determining the final model checking result. The players make moves between configurations in which they try to verify or refute Φ' in s. All possible plays of a game are captured in a game-graph, whose nodes are the elements of the game board and whose edges are the possible moves of the players. The model checking game is solved via a coloring algorithm which colors each node (s, Φ') in the game-graph by T, F,

or ? iff the value of Φ' in s is *true*, *false*, or indefinite iff the winning strategy at (s, Φ') has Player $\forall$, Player $\exists$, or none of the players, respectively. In case the initial node is colored by ?, the game results in a *tie* with an indefinite answer, and we want to refine the abstract model. We can find the reason for the tie by examining the part of the game-graph which has indefinite results. We choose a refinement criterion, which splits abstract configurations so that the new, refined abstract configurations represent smaller subsets of concrete configurations. The game-based model checking algorithm is then used to evaluate the new, refined abstract models, which correspond to the refined abstract configurations. The refinement is applied only to parts of the game-graph from which a tie is possible. Nodes from which there is a winning strategy for one of the players are not changed. Thus, the game-graph of refined abstract models do not grow unnecessarily.

Moreover, in this work we also generalize the definition of abstract variability models [15], so that we obtain more precise abstract models in which more CTL formulae can be proved or disproved. Hence, there are less CTL formulae with indefinite answers. Inspired by Shoham and Grumberg [16] and Larsen and Liu [17], we define an abstract variability model as a generalized MTS (GMTS) in which must-transitions are replaced by *must hyper-transitions*, which connect a single state s to a set of states A. A GMTS contains a must hyper-transition $s \longrightarrow A$, iff for valid variants, there exists a state $s' \in A$ such that $s \longrightarrow s'$ is a transition in that variant. This weakens the standard (under-approximating) condition for must-transitions (they occur in all variants) by allowing the resulting state to be "splitted" in several states (from a set A). In effect, we obtain a more precise abstract model in which more CTL formulae have a definite result (*true* or *false*). We suggest an automatic construction of an initial GMTS and its successive refined abstract models. We adjust for GMTSs the game-based model checking algorithm [13, 14, 15] for checking CTL formulae with 3-valued semantics. If the model checking results in an indefinite value, we find a reason for this result and derive from it how to do the refinement. In this way, we obtain an automatic generalized abstraction-refinement framework that is suitable for both verification and falsification of CTL properties on system families (SPLs).

Finally, we experimentally compare the performances of three approaches for verifying CTL properties of several system families: (1) the generalized abstraction-refinement approach; (2) the regular abstraction-refinement approach (with no hyper-transitions); and (3) the standard CTL lifted model checking algorithm (used as a baseline) [18], which uses no abstraction and is

based on an extended version of NuSMV model checker that is specifically tailored for handling system families.

Let us summarize the contributions of this paper:

- *An automatic abstraction-refinement procedure* for CTL lifted model checking, which relies on game-based model checking algorithm as well as partitioning and abstracting the variability model until a point when precise conclusive results are found for all variants.
- *A proof of correctness and termination of the above procedure* when applied to variability models with finite configuration spaces.
- *A generalized definition of abstract variability models* by introducing the notion of must hyper-transitions.
- *An automatic generalized abstraction-refinement procedure* for CTL lifted model checking, which uses generalized abstract variability models.
- *Experimental evaluation* of the above abstraction-refinement procedures for CTL lifted model checking, which shows scalability gains against the traditional unabstracted CTL lifted model checking based on an extended version of NuSMV model checker [18].
- *Extension* of the above abstraction-refinement procedures to handle μ-calculus properties.

This work is an extended and revised version of [15]. We make the following extensions here: (1) We generalize the definition of abstract variability models, thus obtaining more precise abstract models; (2) We integrate the newly defined generalized abstract variability models into an abstraction-refinement procedure for CTL lifted model checking; (3) We provide additional explanations and formal proofs for all main results in the work, the old and new alike; (4) We expand and elaborate the examples as well as the discussion on how this approach works; (5) We expand the evaluation of this approach by implementing the new generalized abstraction-refinement procedure, considering more properties, and extending the performance results; (6) We show how our abstraction-refinement procedure can be extended to handle μ-calculus.

The paper is organized as follows. The basics of CTL lifted model checking as well as CTL abstract lifted model checking are explained in Section 2. In Section 3 we define the game-based CTL abstract lifted model checking,

while in Section 4 we integrate this algorithm into an abstraction-refinement framework by defining a suitable notion of refinement. In Section 5 we define the generalized abstract variability models, while in Section 6 we integrate them into a generalized abstraction-refinement framework. The implementation and evaluation are presented in Section 7. We extend our approach with μ-calculus properties in Section 8. Finally, we discuss the related work and conclude.

2. Background

In this section, we present the background for variability models used to represent system families, for their abstractions, and for semantics of CTL.

2.1. System Families

Definition. Let $\mathbb{F} = \{A_1, \ldots, A_n\}$ be a finite set of Boolean variables representing the features available in a system family. A specific subset of features, $k \subseteq \mathbb{F}$, known as *configuration*, specifies a *variant* of a system family. We assume that only a subset $\mathbb{K} \subseteq 2^{\mathbb{F}}$ of configurations are *valid*. An alternative representation of configurations is based upon propositional formulae. Each configuration $k \in \mathbb{K}$ can be represented by a formula: $k(A_1) \wedge \ldots \wedge k(A_n)$, where $k(A_i) = A_i$ if $A_i \in k$, and $k(A_i) = \neg A_i$ if $A_i \notin k$ for $1 \leq i \leq n$. We will use both representations interchangeably.

We use *transition systems* (TS) to describe behaviors of single systems. A *transition system* is a tuple $\mathcal{T} = (S, I, \mathit{trans}, AP, L)$, where S is a set of states; $I \subseteq S$ is a set of initial states; $\mathit{trans} \subseteq S \times S$ is a transition relation which is *total*, so that for each state there is an outgoing transition; AP is a set of atomic propositions; and $L : S \to 2^{AP}$ is a labelling function specifying which atomic propositions hold in a state. We write $s_1 \longrightarrow s_2$ whenever $(s_1, s_2) \in \mathit{trans}$. A *path* (behaviour) of a TS $\mathcal{T}$ is an *infinite* sequence $\rho = s_0 s_1 s_2 \ldots$ with $s_0 \in I$ such that $s_i \longrightarrow s_{i+1}$ for all $i \geq 0$. The *semantics* of the TS $\mathcal{T}$, denoted as $[\![\mathcal{T}]\!]_{TS}$, is the set of its paths.

A *featured transition system* (FTS) represents a compact model, which describes the behavior of a whole family of systems in a single monolithic description. Their transitions are guarded by a *presence condition* that identifies the variants they belong to. The presence conditions ψ are drawn from the set of feature expressions, $\mathit{FeatExp}(\mathbb{F})$, which are propositional logic formulae over $\mathbb{F}$:

$$\psi ::= \mathit{true} \mid A \in \mathbb{F} \mid \neg\psi \mid \psi_1 \wedge \psi_2$$

We write $[\![\psi]\!]$ for the set of configurations that satisfy ψ, i.e. $k \in [\![\psi]\!]$ iff $k \models \psi$.

A *featured transition system* (FTS) is a tuple $\mathcal{F}=(S,I,trans,AP,L,\mathbb{F},\mathbb{K},\delta)$, where $(S, I, trans, AP, L)$ form a TS; $\mathbb{F}$ is a set of available features; $\mathbb{K}$ is a set of valid configurations; and $\delta : trans \to FeatExp(\mathbb{F})$ is a total function decorating transitions with presence conditions (feature expressions). The *projection* of an FTS $\mathcal{F}$ to a configuration $k \in \mathbb{K}$, denoted as $\pi_k(\mathcal{F})$, is the TS $(S, I, trans', AP, L)$, where $trans' = \{t \in trans \mid k \models \delta(t)\}$. We lift the definition of *projection* to sets of configurations $\mathbb{K}' \subseteq \mathbb{K}$, denoted as $\pi_{\mathbb{K}'}(\mathcal{F})$, by keeping the transitions admitted by at least one of the configurations in $\mathbb{K}'$. That is, $\pi_{\mathbb{K}'}(\mathcal{F})$, is the FTS $(S, I, trans', AP, L, \mathbb{F}, \mathbb{K}', \delta')$, where $trans' = \{t \in trans \mid \exists k \in \mathbb{K}'.k \models \delta(t)\}$ and $\delta' = \delta|_{trans'}$ is the restriction of δ to $trans'$. The *semantics* of an FTS $\mathcal{F}$, denoted as $[\![\mathcal{F}]\!]_{FTS}$, is the union of paths (behaviours) of the projections on all valid variants $k \in \mathbb{K}$, i.e. $[\![\mathcal{F}]\!]_{FTS} = \cup_{k\in\mathbb{K}}[\![\pi_k(\mathcal{F})]\!]_{TS}$. Moreover, we have $[\![\pi_{\mathbb{K}'}(\mathcal{F})]\!]_{FTS} = \cup_{k\in\mathbb{K}'}[\![\pi_k(\mathcal{F})]\!]_{TS}$.

Example 1. *Throughout this paper, we will use a beverage vending machine as a running example [4]. Figure 1 shows the FTS $\mathcal{F}_1$, which has two features and each of them is assigned an identifying letter and a color. The features are:* `CancelPurchase` *(c, in brown), for canceling a purchase after a coin is entered; and* `FreeDrinks` *(f, in blue) for offering free drinks. Each transition is labeled by a* feature expression. *For instance, the transition $s_0 \xrightarrow{f} s_2$ is included in variants where the feature f is enabled. For clarity, we omit to write the presence condition true in transitions. There are two atomic propositions $a, r \in AP$, such that $a, r \in L(s_2)$ and $r \in L(s_1)$, whereas $a, r \notin L(s_0)$. Note that proposition r holds in states where a purchase is ordered in the machine, whereas proposition a holds in states where a drink is served by the machine.*

By combining various features, a number of variants of $\mathcal{F}_1$ can be obtained. The set of valid configurations is: $\mathbb{K}_1 = \{\emptyset, \{c\}, \{f\}, \{c, f\}\}$ (or, equivalently $\mathbb{K}_1 = \{\neg c \wedge \neg f, c \wedge \neg f, \neg c \wedge f, c \wedge f\}$). Figure 2 shows a basic version of $\mathcal{F}_1$, described by the configuration: $\emptyset$ (or, as formula $\neg c \wedge \neg f$). This machine takes a coin, serves a drink, and then waits for another order again. □

CTL Properties. For specifying system properties, we consider the logic CTL (e.g., see [11, 19, 10]). CTL state formulae Φ are defined by:

$$\Phi ::= true \mid false \mid l \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid A\phi \mid E\phi$$
$$\phi ::= \bigcirc\Phi \mid \Phi_1 \mathsf{U} \Phi_2 \mid \Phi_1 \mathsf{V} \Phi_2$$

where $l \in Lit = AP \cup \{\neg a \mid a \in AP\}$ and ϕ is a CTL path formulae. The path formula $\bigcirc\Phi$ can be read as "from the next state Φ", $\Phi_1 \mathsf{U} \Phi_2$ can be read

as "Φ_1 until Φ_2", whereas $\Phi_1 \mathsf{V} \Phi_2$ can be read as "Φ_2 while not Φ_1" (where Φ_1 may never hold).

Note that CTL state formulae Φ are given in negation normal form ($\neg$ is applied only to atomic propositions). This facilitates the definition of universal and existential subsets of CTL in which the only allowed path quantifiers are A (always) and E (exists), respectively. Given $\Phi \in \text{CTL}$, we consider $\neg\Phi$ to be the equivalent CTL formula given in negation normal form. To ensure that every CTL formula is equivalent to a formula in negation normal form, for each operator the corresponding dual operator is necessary. We have that $\wedge$ and $\vee$ are dual, $\bigcirc$ is dual to itself, U and V are dual. For example, we have the duality law: $\neg\forall \bigcirc \Phi \equiv \exists \bigcirc \neg\Phi$.

The concrete semantics of CTL over TSs is standard [19, 10]. We write $[\mathcal{T}, s \models \Phi] = tt$ (resp., ff) to denote that the CTL state formula Φ is true (resp., false) in the state s of $\mathcal{T}$, whereas $[\mathcal{T}, \rho \models \phi] = tt$ (resp., ff) has the same meaning for the CTL path formula ϕ over the path ρ of $\mathcal{T}$. $[\mathcal{T}, s \models \Phi]$ is defined as:

(1) $[\mathcal{T}, s \models a] = tt$ iff $a \in L(s)$, $\quad [\mathcal{T}, s \models \neg a] = tt$ iff $a \notin L(s)$

(2) $[\mathcal{T}, s \models \Phi_1 \wedge \Phi_2] = tt$ iff $[\mathcal{T}, s \models \Phi_1] = tt$ and $[\mathcal{T}, s \models \Phi_2] = tt$,
$[\mathcal{T}, s \models \Phi_1 \vee \Phi_2] = tt$ iff $[\mathcal{T}, s \models \Phi_1] = tt$ or $[\mathcal{T}, s \models \Phi_2] = tt$

(3) $[\mathcal{T}, s \models A\phi] = tt$ iff $\forall \rho \in [\![\mathcal{T}]\!]^s_{\text{TS}}. [\mathcal{T}, \rho \models \phi] = tt$;
$[\mathcal{T}, s \models E\phi] = tt$ iff $\exists \rho \in [\![\mathcal{T}]\!]^s_{\text{TS}}. [\mathcal{T}, \rho \models \phi] = tt$

where $[\![\mathcal{T}]\!]^s_{TS}$ denotes the set of all paths starting in state s. $[\mathcal{T}, \rho \models \phi]$ is:

(4) $[\mathcal{T}, \rho \models \bigcirc\Phi] = tt$ iff $[\mathcal{T}, \rho_1 \models \Phi]$,
$[\mathcal{T}, \rho \models (\Phi_1 \mathsf{U} \Phi_2)] = tt$ iff $\exists i \geq 0. \Big([\mathcal{T}, \rho_i \models \Phi_2] = tt \wedge (\forall 0 \leq j < i. [\mathcal{T}, \rho_j \models \Phi_1] = tt)\Big)$,
$[\mathcal{T}, \rho \models (\Phi_1 \mathsf{V} \Phi_2)] = tt$ iff $\forall i \geq 0. \Big(\forall 0 \leq j < i. [\mathcal{T}, \rho_j \models \Phi_1] = ff \implies [\mathcal{T}, \rho_i \models \Phi_2] = tt\Big)$

where $\rho_i = s_i$ denotes the i-th state of the path $\rho = s_0 s_1 s_2 \ldots$.

We say that $\mathcal{T}$ satisfies a CTL formula Φ, denoted $[\mathcal{T} \models \Phi] = tt$, iff $\forall s_0 \in I. [\mathcal{T}, s_0 \models \Phi] = tt$. Otherwise, $\mathcal{T}$ refutes Φ, denoted $[\mathcal{T} \models \Phi] = ff$.

We say that an FTS $\mathcal{F}$ satisfies a CTL formula Φ, written $[\mathcal{F} \models \Phi] = tt$, iff all its valid variants satisfy the formula, i.e. $\forall k \in \mathbb{K}. [\pi_k(\mathcal{F}) \models \Phi] = tt$. Otherwise, we say $\mathcal{F}$ does not satisfy Φ, written $[\mathcal{F} \models \Phi] = ff$. In this case,

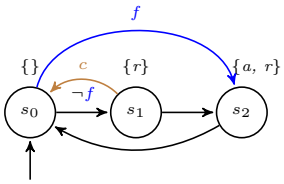

Figure 1: FTS $\mathcal{F}_1$.

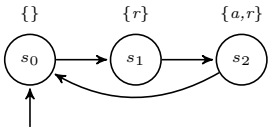

Figure 2: TS $\pi_{\neg c \wedge \neg f}(\mathcal{F}_1)$

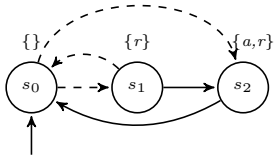

Figure 3: MTS $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$

we also want to determine the non-empty set of violating variants $\mathbb{K}' \subseteq \mathbb{K}$, such that $\forall k' \in \mathbb{K}'. [\pi_{k'}(\mathcal{F}) \models \Phi] = ff$ and $\forall k \in \mathbb{K} \backslash \mathbb{K}'. [\pi_k(\mathcal{F}) \models \Phi] = tt$.

Example 2. *Recall FTS $\mathcal{F}_1$ from Fig. 1. Consider the property $\Phi_1 = A(\neg a U a)$, which states that from the initial state, every path will eventually reach the state where a holds. Note that $[\mathcal{F}_1 \models \Phi_1] = ff$. E.g., if feature c is enabled and f is disabled, a counter-example where the state s_2 that satisfies a is never reached is: $s_0 \to s_1 \to s_0 \to \ldots$. However, $[\pi_{[\![\neg c \vee f]\!]}(\mathcal{F}_1) \models \Phi_1] = tt$.*

Consider the property $\Phi_2 = E(\neg r U r)$, which describes a situation where in the initial state there exists a path that will eventually reach s_1 or s_2 that satisfy r. Note that $[\mathcal{F}_1 \models \Phi_2] = tt$, since for all variants there is a path from the state s_0 to either s_1 or s_2. □

2.2. Abstraction

Definition. We use *modal transition systems* (MTSs) [20] to represent abstract variability models of FTSs that preserve full CTL. A *modal transition system* is a tuple $\mathcal{M} = (S, I, trans^{\text{may}}, trans^{\text{must}}, AP, L)$, where $trans^{\text{may}} \subseteq S \times S$ describes may transitions of $\mathcal{M}$; and $trans^{\text{must}} \subseteq S \times S$ describes must transitions of $\mathcal{M}$, such that $trans^{\text{may}}$ is total and $trans^{\text{must}} \subseteq trans^{\text{may}}$. The intuition behind the inclusion $trans^{must} \subseteq trans^{may}$ is that transitions that are necessarily true ($trans^{must}$) are also possibly true ($trans^{may}$). A *may-path* in $\mathcal{M}$ is a path with all its transitions in $trans^{\text{may}}$; whereas a *must-path* in $\mathcal{M}$ is a maximal sequence with all its transitions in $trans^{\text{must}}$, which cannot be extended with any other transition from $trans^{\text{must}}$. Note that since $trans^{\text{must}}$ is not necessarily total, must-paths can be finite. We use $[\![\mathcal{M}]\!]^{\text{may}}_{MTS}$ (resp., $[\![\mathcal{M}]\!]^{\text{must}}_{MTS}$) to denote the set of all may-paths (resp., must-paths) in $\mathcal{M}$.

CTL Properties. We define the 3-valued semantics of CTL over an MTSs $\mathcal{M}$ [21] slightly differently from the 2-valued semantics for TSs. We define $[\mathcal{M}, s \models^3 \Phi]$ for CTL state formulae Φ, and similarly $[\mathcal{M}, \rho \models^3 \phi]$ for CTL path formulae ϕ. Intuitively, we examine the truth of a formula of the form

$A\phi$ along all may-paths, whereas its falsity is shown by a single must path. On the other hand, for a formula of the form $E\phi$ the opposite is done: its truth is shown by a single must-path, whereas its falsity along all may-paths. More formally, we have ($\mathcal{M}$ is omitted when clear from context):

$$\textbf{(1)}\ [s \models^3 a] = \begin{cases} tt, & \text{if } a \in L(s) \\ ff, & \text{if } a \notin L(s) \end{cases}, \quad [s \models^3 \neg a] = \begin{cases} tt, & \text{if } a \notin L(s) \\ ff, & \text{if } a \in L(s) \end{cases}$$

$$\textbf{(2)}\ [s \models^3 \Phi_1 \wedge \Phi_2] = \begin{cases} tt, & \text{if } [s \models^3 \Phi_1] = tt \wedge [s \models^3 \Phi_2] = tt \\ ff, & \text{if } [s \models^3 \Phi_1] = ff \vee [s \models^3 \Phi_2] = ff \\ \bot, & \text{otherwise} \end{cases}$$

$$\textbf{(3)}\ [s \models^3 A\phi] = \begin{cases} tt, & \text{if } \forall \rho \in [\![\mathcal{M}]\!]^{\text{may},s}_{MTS}.\, [\rho \models^3 \phi] = tt \\ ff, & \text{if } \exists \rho \in [\![\mathcal{M}]\!]^{\text{must},s}_{MTS}.\, [\rho \models^3 \phi] = ff \\ \bot, & \text{otherwise} \end{cases}$$

$$[s \models^3 E\phi] = \begin{cases} tt, & \text{if } \exists \rho \in [\![\mathcal{M}]\!]^{\text{must},s}_{MTS}.\, [\rho \models^3 \phi] = tt \\ ff, & \text{if } \forall \rho \in [\![\mathcal{M}]\!]^{\text{may},s}_{MTS}.\, [\rho \models^3 \phi] = ff \\ \bot, & \text{otherwise} \end{cases}$$

$$\textbf{(4)}\ [\rho \models^3 \bigcirc \Phi] = \begin{cases} [\rho_1 \models^3 \Phi], & \text{if } |\rho| > 1 \\ \bot, & \text{otherwise} \end{cases}$$

$$[\rho \models^3 (\Phi_1 \mathsf{U} \Phi_2)] = \begin{cases} tt, & \text{if } \exists i \leq |\rho|.\Big([\rho_i \models^3 \Phi_2] = tt \wedge (\forall j < i.[\rho_j \models^3 \Phi_1] = tt)\Big) \\ ff, & \text{if } \begin{array}{l} \forall i \leq |\rho|.\Big(\forall j < i.[\rho_j \models^3 \Phi_1] \neq ff \Rightarrow [\rho_i \models^3 \Phi_2] = ff\Big) \\ \wedge\ \forall i \geq 0.[\rho_i \models^3 \Phi_1] \neq ff \Rightarrow |\rho| = \infty \end{array} \\ \bot, & \text{otherwise} \end{cases}$$

$$[\rho \models^3 (\Phi_1 \mathsf{V} \Phi_2)] = \begin{cases} tt, & \text{if } \begin{array}{l} \forall 0 \leq i \leq |\rho|.\Big(\forall j < i.[\rho_j \models^3 \Phi_1] \neq tt \Longrightarrow [\rho_i \models^3 \Phi_2] = tt\Big) \\ \wedge\ \forall i \geq 0.[\rho_i \models^3 \Phi_1] \neq tt \Longrightarrow |\rho| = \infty \end{array} \\ ff, & \text{if } \exists i \geq 0.\Big([\rho_i \models^3 \Phi_2] = ff \wedge (\forall j < i.\,[\rho_j \models^3 \Phi_1] = ff)\Big) \\ \bot, & \text{otherwise} \end{cases}$$

where $[\![\mathcal{M}]\!]^{\text{may},s}_{MTS}$ (resp., $[\![\mathcal{M}]\!]^{\text{must},s}_{MTS}$) denotes the set of all may-paths (must-paths) starting in the state s, and $|\rho|$ denotes the length of ρ.

Construction of an Abstract Model. We start working with Galois connections [22, 7, 9] between Boolean complete lattices of feature expressions, and then induce a notion of abstraction of FTSs. The Boolean complete lattice of feature expressions (propositional formulae over $\mathbb{F}$) is: $(\mathit{FeatExp}(\mathbb{F})_{/\equiv}, \models, \vee, \wedge, \mathit{true}, \mathit{false}, \neg)$, where the elements of the domain $\mathit{FeatExp}(\mathbb{F})_{/\equiv}$ are equivalence classes of propositional formulae $\psi \in \mathit{FeatExp}(\mathbb{F})$ obtained by quotienting by the semantic equivalence $\equiv$.

The (over-approximating) *join abstraction*, $\boldsymbol{\alpha}^{\text{join}}$, replaces each feature expression ψ with *true* if there exists at least one configuration from $\mathbb{K}$ that satisfies ψ. The abstract set of features is empty: $\boldsymbol{\alpha}^{\text{join}}(\mathbb{F}) = \emptyset$, and abstract set of configurations is a singleton: $\boldsymbol{\alpha}^{\text{join}}(\mathbb{K}) = \{\mathit{true}\}$. The abstraction and concretization functions between $\mathit{FeatExp}(\mathbb{F})$ and $\mathit{FeatExp}(\emptyset)$ are:

$$\boldsymbol{\alpha}^{\text{join}}(\psi) = \begin{cases} \mathit{true} & \text{if } \exists k \in \mathbb{K}.k \models \psi \\ \mathit{false} & \text{otherwise} \end{cases} \qquad \boldsymbol{\gamma}^{\text{join}}(\psi) = \begin{cases} \mathit{true} & \text{if } \psi \text{ is } \mathit{true} \\ \bigvee_{k \in 2^{\mathbb{F}} \setminus \mathbb{K}} k & \text{if } \psi \text{ is } \mathit{false} \end{cases}$$

which form a Galois connection [7]. In this way, we obtain a single abstract model (variant) that includes all transitions occurring in some concrete variants. The information about which transitions are associated with which variants is lost, thus causing a precision loss in the abstract model.

The (under-approximating) *dual join abstraction*, $\widetilde{\boldsymbol{\alpha}^{\text{join}}}$, replaces each feature expression ψ with *true* if all configurations from $\mathbb{K}$ satisfy ψ. The abstraction and concretization functions between $\mathit{FeatExp}(\mathbb{F})$ and $\mathit{FeatExp}(\emptyset)$, forming a Galois connection [9], are defined as follows [22]: $\widetilde{\boldsymbol{\alpha}^{\text{join}}} = \neg \circ \boldsymbol{\alpha}^{\text{join}} \circ \neg$ and $\widetilde{\boldsymbol{\gamma}^{\text{join}}} = \neg \circ \boldsymbol{\gamma}^{\text{join}} \circ \neg$, so:

$$\widetilde{\boldsymbol{\alpha}^{\text{join}}}(\psi) = \begin{cases} \mathit{true} & \text{if } \forall k \in \mathbb{K}.k \models \psi \\ \mathit{false} & \text{otherwise} \end{cases} \qquad \widetilde{\boldsymbol{\gamma}^{\text{join}}}(\psi) = \begin{cases} \bigwedge_{k \in 2^{\mathbb{F}} \setminus \mathbb{K}} (\neg k) & \text{if } \psi \text{ is } \mathit{true} \\ \mathit{false} & \text{if } \psi \text{ is } \mathit{false} \end{cases}$$

In this way, we obtain a single abstract model (variant) that includes only those transitions that occur in all concrete variants.

Given the FTS $\mathcal{F} = (S, I, \mathit{trans}, AP, L, \mathbb{F}, \mathbb{K}, \delta)$, we define MTS $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) = (S, I, \mathit{trans}^{\text{may}}, \mathit{trans}^{\text{must}}, AP, L)$ to be its *abstraction*, where $\mathit{trans}^{\text{may}} = \{t \in \mathit{trans} \mid \boldsymbol{\alpha}^{\text{join}}(\delta(t)) = \mathit{true}\}$, $\mathit{trans}^{\text{must}} = \{t \in \mathit{trans} \mid \widetilde{\boldsymbol{\alpha}^{\text{join}}}(\delta(t)) = \mathit{true}\}$. Note that may transitions describe the behaviour that is possible in some variant of the concrete FTS $\mathcal{F}$, but not need be realized in the other variants; whereas must transitions describe behaviour that has to be present in all variants of $\mathcal{F}$.

Example 3. *Recall FTS $\mathcal{F}_1$ of Fig. 1. Figure 3 shows the MTS $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$, where must-transitions are denoted by solid lines, while may-transitions by dashed lines. The allowed (may) part of the behavior of $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ includes transitions that are associated with the optional features c and f in $\mathcal{F}_1$, and the required (must) part includes transitions with presence condition true.* □

The 3-valued CTL semantics of MTS $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})$ is *sound* in the sense that it preserves both satisfaction (*tt*) and refutation (*ff*) of a formula from the abstract model $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})$ to the concrete one $\mathcal{F}$. However, if the truth value of a formula in the abstract model is $\bot$, then its value over the concrete model is not known. We use Lemma 4 from [9] to prove soundness (Theorem 5).

Lemma 4 ([9])**. (i)** *Let $k \in \mathbb{K}$ and $\rho \in [\![\pi_k(\mathcal{F})]\!]_{TS}$. Then, $\rho \in [\![\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})]\!]^{may}_{MTS}$.*

(ii) *Let $\rho \in [\![\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})]\!]^{must}_{MTS}$. Then, $\rho \in [\![\pi_k(\mathcal{F})]\!]_{TS}$ for all $k \in \mathbb{K}$.*

Theorem 5 (Preservation results)**.** *For every $\Phi \in CTL$, we have:*

(1) $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 \Phi] = tt \implies [\mathcal{F} \models \Phi] = tt$, *and* $\forall k \in \mathbb{K}.[\pi_k(\mathcal{F}) \models \Phi] = tt$.

(2) $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 \Phi] = ff \implies [\mathcal{F} \models \Phi] = ff$, *and* $\forall k \in \mathbb{K}.[\pi_k(\mathcal{F}) \models \Phi] = ff$.

Proof. *By induction on the structure of Φ. All cases except A and E quantifiers are straightforward.*
*Consider (**1**):* $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 \Phi] = tt \implies [\mathcal{F} \models \Phi] = tt$.

*Case $\Phi = A\phi$. To prove (**1**), we proceed by contraposition. Assume that $[\mathcal{F} \models A\phi] \neq tt$. Then, there exists a configuration $k \in \mathbb{K}$ and a path $\rho \in [\![\pi_k(\mathcal{F})]\!]_{TS}$, such that $[\pi_k(\mathcal{F}), \rho \models \phi] \neq tt$, i.e. $[\pi_k(\mathcal{F}), \rho \models \neg\phi] = tt$. By Lemma 4(i), we have that $\rho \in [\![\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})]\!]^{may}_{MTS}$. Thus, $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}), \rho \models^3 \phi] \neq tt$, and so $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 A\phi] \neq tt$ by definition.*

*Case $\Phi = E\phi$. To prove (**1**), we assume $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 E\phi] = tt$. This means that there exists a path $\rho \in [\![\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})]\!]^{must}_{MTS}$ such that $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}), \rho \models^3 \phi] = tt$. By Lemma 4(ii), we have that for all $k \in \mathbb{K}$, it holds $\rho \in [\![\pi_k(\mathcal{F})]\!]_{TS}$. Therefore, $[\pi_k(\mathcal{F}) \models E\phi] = tt$ for all $k \in \mathbb{K}$, and so $[\mathcal{F} \models E\phi] = tt$.*

*Consider (**2**):* $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 \Phi] = ff \implies [\mathcal{F} \models \Phi] = ff$.

*Case $\Phi = A\phi$. To prove (**2**), we assume $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 A\phi] = ff$. This means that there exists a path $\rho \in [\![\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})]\!]^{must}_{MTS}$ such that $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}), \rho \models^3 \phi] = ff$. By Lemma 4(ii), we have that for all $k \in \mathbb{K}$, it holds $\rho \in [\![\pi_k(\mathcal{F})]\!]_{TS}$. Therefore, $[\pi_k(\mathcal{F}) \models A\phi] = ff$ for all $k \in \mathbb{K}$, and so $[\mathcal{F} \models A\phi] = ff$.*

Case $\Phi = E\phi$. *To prove (**2**), we proceed by contraposition. Assume that* $[\mathcal{F} \models E\phi] \neq \mathit{ff}$. *Then, there exists a configuration* $k \in \mathbb{K}$ *and a path* $\rho \in [\![\pi_k(\mathcal{F})]\!]_{TS}$, *such that* $[\pi_k(\mathcal{F}), \rho \models \phi] \neq \mathit{ff}$, *i.e.* $[\pi_k(\mathcal{F}), \rho \models \phi] = \mathit{tt}$. *By Lemma 4(i), we have that* $\rho \in [\![\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})]\!]^{may}_{MTS}$. *Thus,* $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}), \rho \models^3 \phi] \neq \mathit{ff}$, *and so* $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 E\phi] \neq \mathit{ff}$ *by definition.* □

The problem of evaluating $[\mathcal{F} \models \Phi]$ can be reduced to a number of smaller problems by partitioning the configuration space $\mathbb{K}$. Let the subsets $\mathbb{K}_1, \mathbb{K}_2, \ldots, \mathbb{K}_n$ form a *partition* of $\mathbb{K}$. Then, $[\mathcal{F} \models \Phi] = \mathit{tt}$ iff $[\pi_{\mathbb{K}_i}(\mathcal{F}) \models \Phi] = \mathit{tt}$ for all $i = 1,\ldots,n$. Also, $[\mathcal{F} \models \Phi] = \mathit{ff}$ iff $[\pi_{\mathbb{K}_j}(\mathcal{F}) \models \Phi] = \mathit{ff}$ for some $1 \leq j \leq n$.

Corollary 6. *Let* $\mathbb{K}_1, \mathbb{K}_2, \ldots, \mathbb{K}_n$ *form a* partition *of* $\mathbb{K}$.

(1) *If* $[\boldsymbol{\alpha}^{\text{join}}(\pi_{\mathbb{K}_1}(\mathcal{F})) \models \Phi] = \mathit{tt} \wedge \ldots \wedge [\boldsymbol{\alpha}^{\text{join}}(\pi_{\mathbb{K}_n}(\mathcal{F})) \models \Phi] = \mathit{tt}$, *then* $[\mathcal{F} \models \Phi] = \mathit{tt}$.

(2) *If* $[\boldsymbol{\alpha}^{\text{join}}(\pi_{\mathbb{K}_j}(\mathcal{F})) \models \Phi] = \mathit{ff}$ *for some* $1 \leq j \leq n$, *then* $[\mathcal{F} \models \Phi] = \mathit{ff}$. *Moreover, it holds* $[\pi_k(\mathcal{F}) \models \Phi] = \mathit{ff}$ *for all* $k \in \mathbb{K}_j$.

Example 7. *Recall FTS* $\mathcal{F}_1$ *of Fig. 1 and MTS* $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ *of Fig. 3. Consider the properties* $\Phi_1 = A(\neg a \mathsf{U} a)$ *and* $\Phi_2 = E(\neg r \mathsf{U} r)$ *introduced in Example 2. We have* $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1) \models^3 \Phi_1] = \bot$, *since (1) there is a may-path in* $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ *where* s_2 *is never reached:* $s_0 \to s_1 \to s_0 \to \ldots$, *and (2) there is no must-path that violates* Φ_1. *We also have* $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1) \models^3 \Phi_2] = \bot$, *since (1) there is no must-path in* $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ *that reaches* s_2 *or* s_1 *from* s_0, *and (2) there is a may-path that satisfies* Φ_2. *So we cannot conclude whether* Φ_1 *and* Φ_2 *are satisfied or not by* $\mathcal{F}_1$, *using the abstract model* $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$. □

In summary, abstract variability models are conservative for definite (*tt* and *ff*) verdicts. Whenever an "indefinite" ($\bot$) verdict occurs as in Example 7, a refinement is needed to make abstract models more precise until a definite verdict is obtained. In the following sections we will define such refinement.

3. Abstract lifted model checking

The 3-valued model checking game on an MTS $\mathcal{M}$ with state set S, a state $s \in S$, and a CTL formula Φ, as introduced in [13, 14], is played by Player $\forall$ and Player $\exists$ in order to evaluate Φ in the state s of $\mathcal{M}$. The goal of Player $\forall$ is either to refute Φ on $\mathcal{M}$ or to prevent Player $\exists$ from verifying it. The goal of Player $\exists$ is either to verify Φ on $\mathcal{M}$ or to prevent Player $\forall$

from refuting it. The game is played on a *game board*, which is the Cartesian product $S \times sub(\Phi)$ of the sets of states S and subformulae of Φ, where $sub(\Phi)$ is defined as:

if $\Phi \in \{true, false, l\}$, then $sub(\Phi) = \{\Phi\}$;
if $\Phi \in \{Æ \bigcirc \Phi_1\}$, then $sub(\Phi) = \{\Phi\} \cup sub(\Phi_1)$
if $\Phi \in \{\Phi_1 \wedge \Phi_2, \Phi_1 \vee \Phi_2\}$, then $sub(\Phi) = \{\Phi\} \cup sub(\Phi_1) \cup sub(\Phi_2)$
if $\Phi \in \{Æ(\Phi_1 \mathsf{U} \Phi_2), Æ(\Phi_1 \mathsf{V} \Phi_2)\}$, then $sub(\Phi) = exp(\Phi) \cup sub(\Phi_1) \cup sub(\Phi_2)$

where Æ ranges over both A and E. As a result of expansion equivalence laws [19]: $Æ(\Phi_1 \mathsf{U} \Phi_2) \equiv \Phi_2 \vee (\Phi_1 \wedge Æ \bigcirc \Phi)$ and $Æ(\Phi_1 \mathsf{V} \Phi_2) \equiv \Phi_2 \wedge (\Phi_1 \vee Æ \bigcirc \Phi)$, the expansion $exp(\Phi)$ is defined as: if $\Phi = Æ(\Phi_1 \mathsf{U} \Phi_2)$, then $exp(\Phi) = \{\Phi, \Phi_2 \vee (\Phi_1 \wedge Æ \bigcirc \Phi), \Phi_1 \wedge Æ \bigcirc \Phi, Æ \bigcirc \Phi\}$; if $\Phi = Æ(\Phi_1 \mathsf{V} \Phi_2)$, then $exp(\Phi) = \{\Phi, \Phi_2 \wedge (\Phi_1 \vee Æ \bigcirc \Phi), \Phi_1 \vee Æ \bigcirc \Phi, Æ \bigcirc \Phi\}$.

We can define formally any play of a game using the notion of a *configuration*. Intuitively, a configuration contains a complete description of the current state of a play, and it is given as an element of the game board $S \times sub(\Phi)$. A *single play* from a configuration (s, Φ) is a possibly infinite sequence of configurations $C_0 \rightarrow_{p_0} C_1 \rightarrow_{p_1} C_2 \rightarrow_{p_2} \ldots$, where $C_0 = (s, \Phi)$, $C_i \in S \times sub(\Phi)$, $p_i \in \{\text{Player } \forall, \text{Player } \exists\}$. The subformula in C_i determines which player p_i makes the next move. The possible moves at each configuration are:

(1) $C_i = (s, false)$, $C_i = (s, true)$, $C_i = (s, l)$: the play is finished. Such configurations are called *terminal*.

(2) if $C_i = (s, A \bigcirc \Phi)$, Player $\forall$ chooses a must-transition $s \longrightarrow s'$ (for refutation) or a may-transition $s \longrightarrow s'$ of $\mathcal{M}$ (to prevent satisfaction), and $C_{i+1} = (s', \Phi)$.

(3) if $C_i = (s, E \bigcirc \Phi)$, Player $\exists$ chooses a must-transition $s \longrightarrow s'$ (for satisfaction) or a may-transition $s \longrightarrow s'$ of $\mathcal{M}$ (to prevent refutation), and $C_{i+1} = (s', \Phi)$.

(4) if $C_i = (s, \Phi_1 \wedge \Phi_2)$, then Player $\forall$ chooses $j \in \{1, 2\}$ and $C_{i+1} = (s, \Phi_j)$.

(5) if $C_i = (s, \Phi_1 \vee \Phi_2)$, then Player $\exists$ chooses $j \in \{1, 2\}$ and $C_{i+1} = (s, \Phi_j)$.

(6),(7) if $C_i = (s, Æ(\Phi_1 \mathsf{U} \Phi_2))$, then $C_{i+1} = (s, \Phi_2 \vee (\Phi_1 \wedge Æ \bigcirc Æ(\Phi_1 \mathsf{U} \Phi_2)))$.

(8),(9) if $C_i = (s, Æ(\Phi_1 \mathsf{V} \Phi_2))$, then $C_{i+1} = (s, \Phi_2 \wedge (\Phi_1 \vee Æ \bigcirc Æ(\Phi_1 \mathsf{V} \Phi_2)))$.

The moves (6) − (9) are deterministic, thus any player can make them.

A play is a *maximal play* if it is infinite or ends in a terminal configuration. A play is infinite [12] if there is exactly one subformula of the form $A\mathsf{U}$, $A\mathsf{V}$, $E\mathsf{U}$, or $E\mathsf{V}$ that occurs infinitely often in the play. Such a subformula is called a *witness*. We have the following *winning criteria*:

- Player $\forall$ *wins* a (maximal) play iff in each configuration of the form $C_i = (s, A\bigcirc\Phi)$, Player $\forall$ chooses a move based on must-transitions and one of the following holds: (1) the play is finite and ends in a terminal configuration of the form $C_i = (s, \mathit{false})$ or $C_i = (s, a)$ where $a \notin L(s)$ or $C_i = (s, \neg a)$ where $a \in L(s)$; (2) the play is infinite and the witness is of the form $A\mathsf{U}$ or $E\mathsf{U}$.

- Player $\exists$ *wins* a (maximal) play iff in each configuration of the form $C_i = (s, E\bigcirc\Phi)$, Player $\exists$ chooses a move based on must-transitions and one of the following holds: (1) the play is finite and ends in a terminal configuration of the form $C_i = (s, \mathit{true})$ or $C_i = (s, a)$ where $a \in L(s)$ or $C_i = (s, \neg a)$ where $a \notin L(s)$; (2) the play is infinite and the witness is of the form $A\mathsf{V}$ or $E\mathsf{V}$.

- Otherwise, the play ends in a *tie*.

A (memoryless) *strategy* is a set of rules for a player, telling the player which move to choose in the current configuration. A *winning strategy* from (s, Φ) is a set of rules allowing the player to win every play that starts at (s, Φ) if he plays by the rules. It was shown [13, 14] that the model checking problem of evaluating $[\mathcal{M}, s \models^3 \Phi]$ can be reduced to the problem of finding which player has a winning strategy from (s, Φ).

The algorithm for solving the given 3-valued model checking game [13, 14] consists of two parts. First, it constructs a *game-graph*, then it runs an *algorithm for coloring* the game-graph. The game-graph is $G_{\mathcal{M}\times\Phi} = (N, E)$ where $N \subseteq S \times \mathit{sub}(\Phi)$ is the set of nodes and $E \subseteq N \times N$ is the set of edges. N contains a node for each configuration that was reached during the construction of the game-graph that starts from initial configurations $I \times \{\Phi\}$ in a BFS manner, and E contains an edge for each possible move that was applied. The nodes of the game-graph can be classified as: terminal nodes, $\wedge$-nodes, $\vee$-nodes, $A\bigcirc$-nodes, and $E\bigcirc$-nodes. Similarly, the edges can be classified as: *progress edges*, which originate in $A\bigcirc$ or $E\bigcirc$ nodes and reflect real transitions of the MTS $\mathcal{M}$; and *auxiliary edges*, which are all other edges.

We distinguish two types of progress edges, two types of children, and two types of SCCs (Strongly Connected Components) [1]. *Must-edges* (*may-edges*) are progress edges based on must-transitions (may-transitions) of MTSs. A node n' is a *must-child* (*may-child*) of the node n if there exists a must-edge (may-edge) (n, n'). A *must-SCC* (*may-SCC*) is an SCC in which all progress edges are must-edges (may-edges).

Example 8. *The game-graph for MTS* $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ *and* $\Phi_1 = A(\neg a \mathsf{U} a)$ *is shown in Fig. 4, whereas the game-graph for MTS* $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ *and* $\Phi_2 = E(\neg r \mathsf{U} r)$ *is shown in Fig. 5. The model* $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ *has a single initial state* s_0*, thus* $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)\times\Phi_1}$ *and* $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)\times\Phi_2}$ *have single initial nodes* $(s_0, A(\neg a \mathsf{U} a))$ *and* $(s_0, E(\neg r \mathsf{U} r))$*, respectively. The set of formulae appearing in the non-trivial SCCs of* $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)\times\Phi_1}$ *and* $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)\times\Phi_2}$ *are exactly* $exp(A(\neg a \mathsf{U} a))$ *and* $exp(E(\neg r \mathsf{U} r))$*, respectively.* □

The game-graph is partitioned into its may-Maximal SCCs (may-MSCCs), denoted Q_i's. This partition induces a partial order $\leq$ on the Q_i's, such that edges go out of a set Q_i only to itself or to a smaller set Q_j. The partial order is extended to a total order $\leq$ arbitrarily. The *coloring algorithm* processes the Q_i's according to $\leq$, bottom-up. Let Q_i be the smallest set that is not fully colored. The nodes of Q_i are colored in two phases, as follows.
Phase 1. Apply these rules to all nodes in Q_i until none of them is applicable.

- A terminal node C is colored: by T if Player $\exists$ wins in it (when $C = (s, \mathit{true})$ or $C = (s, a)$ with $a \in L(s)$ or $C = (s, \neg a)$ with $a \notin L(s)$); and by F if Player $\forall$ wins in it (when $C = (s, \mathit{false})$ or $C = (s, a)$ with $a \notin L(s)$ or $C = (s, \neg a)$ with $a \in L(s)$).
- An $A\bigcirc$ node is colored: by T if all its may-children are colored by T; by F if it has a must-child colored by F; by ? if all its must-children are colored by T or ?, and it has a may-child colored by F or ?.
- An $E\bigcirc$ node is colored: by T if it has a must-child colored by T; by F if all its may-children are colored by F; by ? if it has a may-child colored by T or ?, and all its must-children are colored by F or ?.

[1] SCCs of a graph are the equivalence classes of nodes under the "are mutually reachable" relation.

- An $\wedge$-node ($\vee$-node) is colored: by T (F) if both its children are colored by T (F); by F (T) if it has a child that is colored by F (T); by ? if it has a child colored by ? and the other child is colored by ? or T (F).

Phase 2. If after propagation of the rules of Phase 1, there are still nodes in Q_i that remain uncolored, then Q_i must be a non-trivial may-MSCC that has exactly one witness that occurs infinitely often in a play. We consider two cases for the witness: Case U and its dual Case V.

Case U. The witness is of the form $A(\Phi_1 \mathsf{U} \Phi_2)$ or $E(\Phi_1 \mathsf{U} \Phi_2)$.

Phase 2a. Repeatedly color by ? each node in Q_i that satisfies one of the following conditions, until there is no change:
(1) An $A\bigcirc$ node such that all its must-children are colored by T or ?; (2) An $E\bigcirc$ node that has a may-child colored by T or ?; (3) An $\wedge$ node that both its children are colored T or ?; (4) An $\vee$ node that has a child colored by T or ?.
In fact, each node for which the F option is no longer possible according to the rules of Phase 1 is colored by ?.
Phase 2b. Color the remaining nodes in Q_i by F.

Case V. The witness is of the form $A(\Phi_1 \mathsf{V} \Phi_2)$ or $E(\Phi_1 \mathsf{V} \Phi_2)$.

Phase 2a. Repeatedly color by ? each node in Q_i that satisfies one of the following conditions, until there is no change.
(1) An $A\bigcirc$ node that has a may-child colored by F or ?; (2) An $E\bigcirc$ node such that all its must-children are colored by F or ?; (3) An $\wedge$ node that has a child colored F or ?; (4) An $\vee$ node that both its children are colored F or ?.
In fact, each node for which the T option is no longer possible according to the rules of Phase 1 is colored by ?.
Phase 2b. Color the remaining nodes in Q_i by T.

The result of the coloring algorithm is a *3-valued coloring function* $\chi : N \to \{T, F, ?\}$, which reflects the 3-valued semantics of CTL.

Theorem 9 ([13])**.** *Let $\mathcal{M}$ be an MTS and Φ be a CTL formula. For each $n = (s, \Phi') \in G_{\mathcal{M} \times \Phi}$:*

(1) $[\mathcal{M}, s \models^3 \Phi'] = tt$ *iff* $\chi(n) = T$ *iff Player* $\exists$ *has a winning strategy at* n.

(2) $[\mathcal{M}, s \models^3 \Phi'] = ff$ *iff* $\chi(n) = F$ *iff Player* $\forall$ *has a winning strategy at* n.

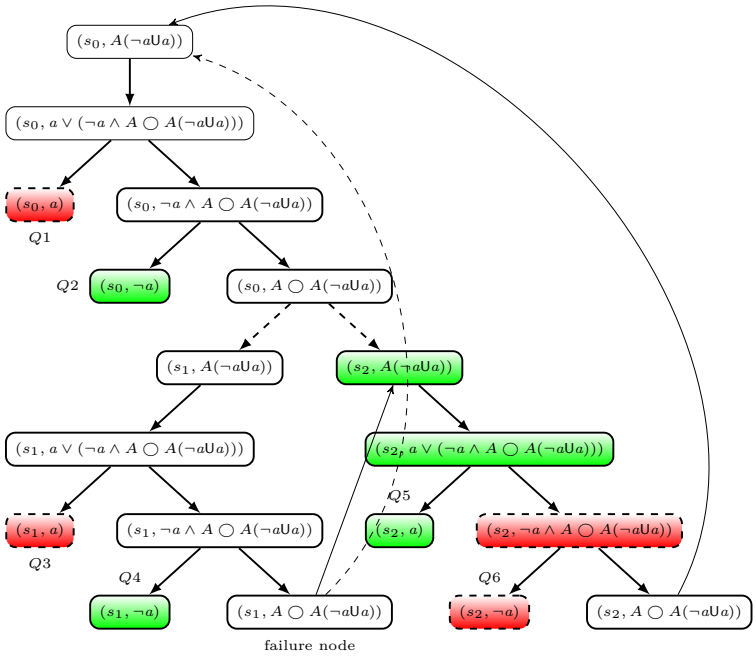

Figure 4: The colored $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)\times\Phi_1}$.

(3) $[\mathcal{M}, s \models^3 \Phi'] = \bot$ *iff* $\chi(n) = ?$ *iff none of the players has a winning strategy at* n.

By Theorem 5 and Theorem 9, we can use the colored game-graph of the MTS $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})$ and Φ to evaluate $[\mathcal{F} \models \Phi]$. If all initial nodes of $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})\times\Phi}$ are colored by T then $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 \Phi] = tt$ and so $[\mathcal{F} \models \Phi] = tt$; if at least one of them is colored by F then $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 \Phi] = ff$ and so $[\mathcal{F} \models \Phi] = ff$. Otherwise, $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 \Phi] = \bot$ and so we do not know the value of $[\mathcal{F} \models \Phi]$.

Corollary 10. *Let* $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})\times\Phi}$ *be a game-graph and* χ *be its coloring function.*

(1) $[\mathcal{F} \models \Phi] = tt$ *iff* $\forall s_0 \in I.\chi((s_0, \Phi)) = T$.

(2) $[\mathcal{F} \models \Phi] = ff$ *iff* $\exists s_0 \in I.\chi((s_0, \Phi)) = F$.

Example 11. *Consider the colored game-graph for MTS* $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ *and* $\Phi_1 = A(\neg a \mathsf{U} a)$ *shown in Fig. 4. Green, red (with dashed borders), and white nodes denote nodes colored by* T, F, *and* $?$, *respectively. The partitions from* Q_1 *to*

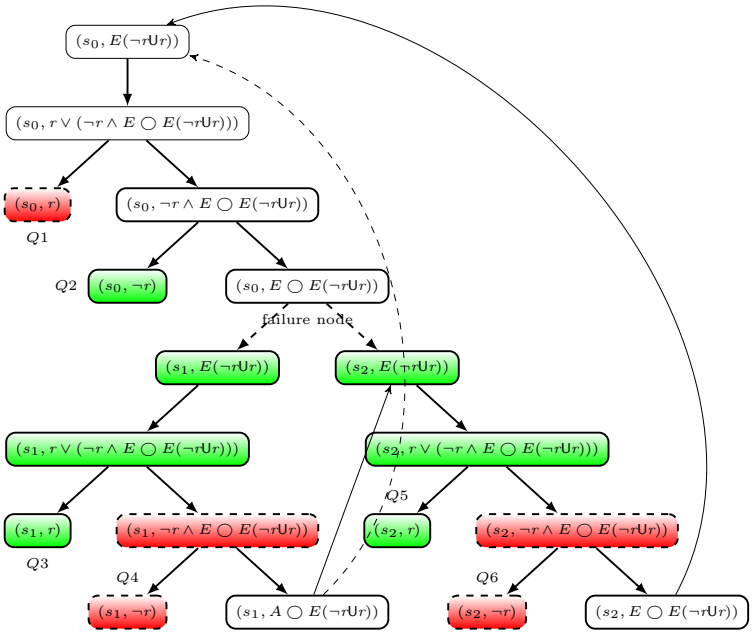

Figure 5: The colored $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)\times\Phi_2}$.

Q_6 consist of a single node shown in Fig. 4, while Q_7 contains all the other nodes. The initial node (s_0, Φ_1) is colored by ?, so we obtain an indefinite answer and we cannot conclude any relation between $\mathcal{F}_1$ and Φ_1.

Consider the colored game-graph for MTS $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ and $\Phi_2 = E(\neg r \mathsf{U} r)$ shown in Fig. 5. The initial node (s_0, Φ_2) is colored by ?, so we again obtain an indefinite answer. □

4. Abstraction-refinement framework

Given an FTS $\mathcal{F}$ with a configuration set $\mathbb{K}$, we show how to exploit the game-graph of the abstract MTS $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})$ in order to do refinement in case that the model checking resulted in an indefinite answer. The refinement consists of two parts. First, we use the information gained by the coloring algorithm of $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})\times\Phi}$ in order to split the single abstract configuration $\mathit{true} \in \boldsymbol{\alpha}^{\text{join}}(\mathbb{K})$ that represents the whole concrete configuration set $\mathbb{K}$, so that the new, refined abstract configurations represent smaller subsets of

concrete configurations. We then construct refined abstract models, using the refined abstract configurations.

There are a failure node and a failure reason associated with an indefinite answer. The goal in the refinement is to find and eliminate the failure reason.

Definition 12. *A node n is a* failure node *if it is colored by ?, whereas none of its children was colored by ? at the time n got colored by the coloring algorithm.*

Such failure node can be seen as the point where the loss of information occurred, so we can use it in the refinement step to change the final model checking result to a definite one.

Lemma 13 ([13])**.** *A failure node is one of the following.*

- *An A◯-node (E◯-node) that has a may-child colored by F (T).*
- *An A◯-node (E◯-node) that was colored during Phase 2a based on an AU (AV) witness, and has a may-child colored by ?.*

Given a failure node $n = (s, \Phi)$, suppose that its may-child is $n' = (s', \Phi'_1)$ as identified in Lemma 13. Then the may-edge from n to n' is considered as *the failure reason*. Thus, we guide the refinement to discard the cause for failure in the hope of changing the model checking result to a definite one. Since the failure reason is a may-transition in the abstract MTS $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})$, it needs to be refined in order to result either in a must transition or no transition at all. Let $s \xrightarrow{\psi} s'$ be the transition in the concrete model $\mathcal{F}$ corresponding to the above (failure) may-transition. We split the configuration space $\mathbb{K}$ into $[\![\psi]\!]$ and $[\![\neg\psi]\!]$ subsets, and we partition $\mathcal{F}$ in $\pi_{[\![\psi]\!]}(\mathcal{F})$ and $\pi_{[\![\neg\psi]\!]}(\mathcal{F})$. Then, we repeat the verification process based on abstract models $\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\psi]\!]}(\mathcal{F}))$ and $\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\neg\psi]\!]}(\mathcal{F}))$. Note that, in the former, $\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\psi]\!]}(\mathcal{F}))$, $s \longrightarrow s'$ becomes a must-transition, while in the latter, $\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\neg\psi]\!]}(\mathcal{F}))$, $s \longrightarrow s'$ is removed. The complete abstraction-refinement procedure is shown in Algorithm 1.

Theorem 14. *The procedure* `Verify`$(\mathcal{F}, \mathbb{K}, \Phi)$ *terminates and is correct.*

Proof. *At the end of an iteration,* `Verify`$(\mathcal{F}, \mathbb{K}, \Phi)$ *either terminates with answers 'tt' or 'ff', or an indefinite result is returned. In the latter case, let ψ be the feature expression guarding the transition in $\mathcal{F}$ that is found as the reason for failure. This (failure) transition occurs as a may-transition in $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})$. We generate $\mathcal{F}_1 = \pi_{[\![\psi]\!]}(\mathcal{F})$ and $\mathbb{K}_1 = \mathbb{K} \cap [\![\psi]\!]$, as well as $\mathcal{F}_2 = \pi_{[\![\neg\psi]\!]}(\mathcal{F})$*

Algorithm 1: `Verify`$(\mathcal{F}, \mathbb{K}, \Phi)$

Input: An FTS $\mathcal{F}$, a configuration set $\mathbb{K}$, and a CTL formula Φ

Output: The value of $[\pi_k(\mathcal{F}) \models \Phi]$ for all $k \in \mathbb{K}$

1 Check by game-based model checking algorithm $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 \Phi]$?;

2 If the result is *tt*, then return $[\pi_k(\mathcal{F}) \models \Phi] = tt$ for all $k \in \mathbb{K}$
If the result is *ff*, then return $[\pi_k(\mathcal{F}) \models \Phi] = ff$ for all $k \in \mathbb{K}$;

3 Otherwise, an indefinite result is obtained in Step (1). Let the may-edge from $n = (s, \Phi_1)$ to $n' = (s', \Phi_1')$ be a failure reason, and let ψ be the feature expression guarding the transition from s to s' in $\mathcal{F}$. We generate $\mathcal{F}_1 = \pi_{[\![\psi]\!]}(\mathcal{F})$ and $\mathcal{F}_2 = \pi_{[\![\neg\psi]\!]}(\mathcal{F})$. Return `Verify`$(\mathcal{F}_1, \mathbb{K} \cap [\![\psi]\!], \Phi)$ $\uplus$ `Verify`$(\mathcal{F}_2, \mathbb{K} \cap [\![\neg\psi]\!], \Phi)$.

and $\mathbb{K}_2 = \mathbb{K} \cap [\![\neg\psi]\!]$. *In the next iteration, we call* `Verify`$(\mathcal{F}_1, \mathbb{K}_1, \Phi)$ *and* `Verify`$(\mathcal{F}_2, \mathbb{K}_2, \Phi)$. *We have that* $\mathbb{K}_1 \subseteq \mathbb{K}$, $\mathbb{K}_2 \subseteq \mathbb{K}$, *and* $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ *contains the (failure) may-transition as a must-transition, while* $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_2)$ *does not contain the (failure) may-transition at all. In this way, we have eliminated the reason for failure in the previous iteration, since only may-transitions can be failure reasons according to Lemma 13. Given that the number of possible updates of the configuration space and the number of may-transitions in abstract models are finite, the number of iterations is also finite.*

If `Verify`$(\pi_{\mathbb{K}'}(\mathcal{F}), \mathbb{K}', \Phi)$ *terminates with answer 'tt' that a property is satisfied (resp., 'ff' that a property is violated) for the variants* $k \in \mathbb{K}'$, *then the answer is correct by Theorem 5, case (**1**) (resp., case (**2**)).* □

Example 15. *We can do a failure analysis on the game-graph* $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1) \times \Phi_1}$ *in Fig. 4. The failure node is* $(s_1, A \bigcirc A(\neg a U a))$ *and the reason is the may-edge* $(s_1, A \bigcirc A(\neg a U a)) \longrightarrow (s_0, A(\neg a U a))$. *The corresponding concrete transition in* $\mathcal{F}_1$ *is* $s_1 \xrightarrow{c} s_0$. *So, we partition the configuration set* $\mathbb{K}_1$ *into subsets* $[\![c]\!]$ *and* $[\![\neg c]\!]$, *and in the next iteration we consider FTSs* $\pi_{[\![c]\!]}(\mathcal{F}_1)$ *and* $\pi_{[\![\neg c]\!]}(\mathcal{F}_1)$.

The failure node for the game-graph $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1) \times \Phi_2}$ *given in Fig. 5 is* $(s_0, E \bigcirc E(\neg r U r))$ *and the reason is the may-edge* $(s_0, E \bigcirc E(\neg r U r)) \longrightarrow (s_1, E(\neg r U r))$. *The corresponding concrete transition in* $\mathcal{F}_1$ *is* $s_0 \xrightarrow{\neg f} s_1$. *So, we partition the configuration space* $\mathbb{K}_1$ *into subsets* $[\![f]\!]$ *and* $[\![\neg f]\!]$, *and in the next second iteration we consider FTSs* $\pi_{[\![f]\!]}(\mathcal{F}_1)$ *and* $\pi_{[\![\neg f]\!]}(\mathcal{F}_1)$. □

The game-based model checking algorithm provides us with a convenient framework to use results from previous iterations and avoid unnecessary calculations. At the end of the i-th iteration of abstraction-refinement, we

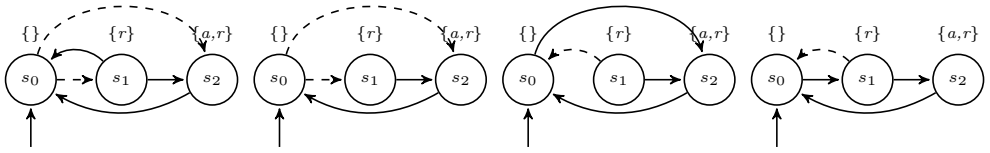

Figure 6: $\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![c]\!]}(\mathcal{F}_1))$ Figure 7: $\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\neg c]\!]}(\mathcal{F}_1))$ Figure 8: $\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![f]\!]}(\mathcal{F}_1))$ Figure 9: $\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\neg f]\!]}(\mathcal{F}_1))$

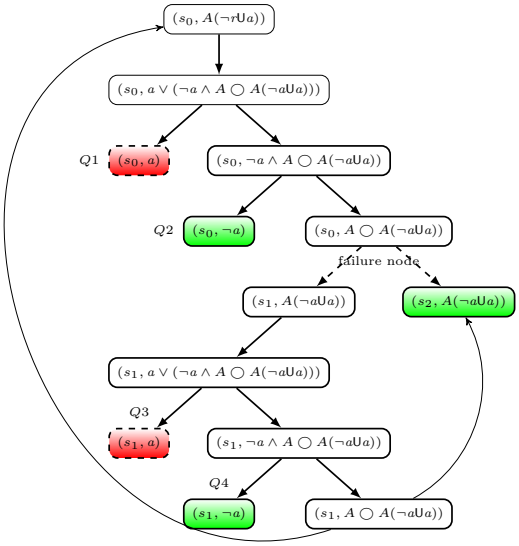

Figure 10: $G_{\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![c]\!]}(\mathcal{F}_1))\times\Phi_1}$.

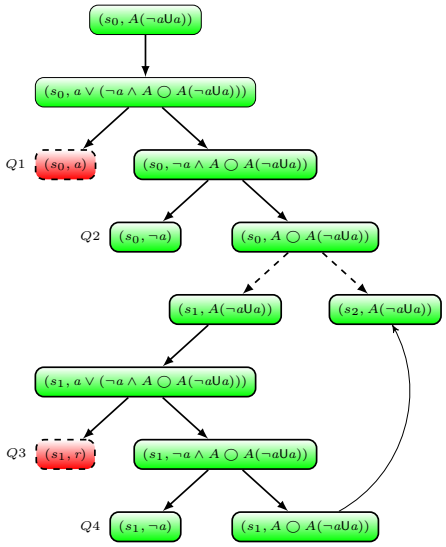

Figure 11: $G_{\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\neg c]\!]}(\mathcal{F}_1))\times\Phi_1}$.

remember those nodes of the game-graph that were colored by definite colors. Let D denote the set of such nodes. Let $\chi_D : D \to \{T, F\}$ be the coloring function that maps each node in D to its definite color. The incremental approach uses this information both in the construction of the game-graph and its coloring in the next iterations. During the construction of a new refined game-graph performed in a BFS manner in the next $i+1$-th iteration, we prune the game-graph in nodes that are from D. When a node $n \in D$ is encountered, we add n to the game-graph and do not continue to construct the game-graph from n onwards. That is, $n \in D$ is considered as terminal node and colored by its previous color. As a result of this pruning, only the reachable sub-graph that was previously colored by ? is refined.

Example 16. *The property* Φ_1 *holds for* $\pi_{[\![\neg c]\!]}(\mathcal{F}_1)$. *We show the model*

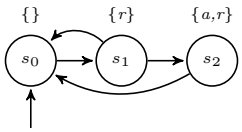

Figure 12: $\pi_{[\![c \wedge \neg f]\!]}(\mathcal{F}_1)$

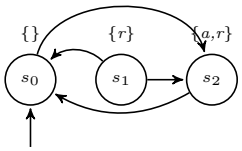

Figure 13: $\pi_{[\![c \wedge f]\!]}(\mathcal{F}_1)$

$\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\neg c]\!]}(\mathcal{F}_1))$ *and the game-graph* $G_{\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\neg c]\!]}(\mathcal{F}_1))\times\Phi_1}$ *in Fig. 7 and Fig. 11 respectively, where the initial node is colored by* T*. Compare the node* $(s_0, A \bigcirc A(\neg a U a))$ *in* $G_{\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\neg c]\!]}(\mathcal{F}_1))\times\Phi_1}$ *(Fig. 11) and* $(s_0, E \bigcirc E(\neg r U r))$ *in* $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)\times\Phi_2}$ *(Fig. 5). All may-children of both nodes are colored by* T*. However, the former node is an* $A\bigcirc$*-node so it will be colored by* T*, whereas the latter is an* $E\bigcirc$*-node so it will be colored by* $?$*.*

On the other hand, we obtain an indefinite answer for $\pi_{[\![c]\!]}(\mathcal{F}_1)$*. The model* $\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![c]\!]}(\mathcal{F}_1))$ *is shown in Fig. 6, whereas the final colored game-graph* $G_{\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![c]\!]}(\mathcal{F}_1))\times\Phi_1}$ *is given in Fig. 10. The failure node is* $(s_0, A \bigcirc A(\neg a U a))$*, and the reason is the may-edge* $(s_0, A \bigcirc A(\neg a U a)) \longrightarrow (s_1, A(\neg a U a))$*. The corresponding concrete transition in* $\pi_{[\![c]\!]}(\mathcal{F}_1)$ *is* $s_0 \xrightarrow{\neg f} s_1$*. So, in the next third iteration we consider FTSs* $\pi_{[\![c \wedge \neg f]\!]}(\mathcal{F}_1)$ *and* $\pi_{[\![c \wedge f]\!]}(\mathcal{F}_1)$*, which are shown in Fig 12 and Fig 13 respectively. Note that* $\pi_{[\![c \wedge \neg f]\!]}(\mathcal{F}_1)$ *and* $\pi_{[\![c \wedge f]\!]}(\mathcal{F}_1)$ *are in fact TSs, so no further variability abstraction can be applied on them. The colored game-graphs* $G_{\pi_{[\![c \wedge \neg f]\!]}(\mathcal{F}_1)\times\Phi_1}$ *and* $G_{\pi_{[\![c \wedge f]\!]}(\mathcal{F}_1)\times\Phi_1}$ *are shown in Fig. 14 and Fig. 15, respectively. The initial node of* $G_{\pi_{[\![c \wedge \neg f]\!]}(\mathcal{F}_1)\times\Phi_1}$ *is colored by* F *in Phase 2b, whereas the initial node of* $G_{\pi_{[\![c \wedge f]\!]}(\mathcal{F}_1)\times\Phi_1}$ *is colored by* T*. Therefore, we conclude that* Φ_1 *is satisfied by the variants* $\{\neg c \wedge \neg f, \neg c \wedge f, c \wedge f\}$*, and* Φ_1 *is violated by the variant* $\{c \wedge \neg f\}$*.*

We need two iterations to conclude that $\Phi_2 = E(\neg r U r)$ *is satisfied by all variants in* $\mathbb{K}_1$*. We show the abstract models* $\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![f]\!]}(\mathcal{F}_1))$ *and* $\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\neg f]\!]}(\mathcal{F}_1))$ *in Fig. 8 and Fig. 9 respectively. The game-graphs* $G_{\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![f]\!]}(\mathcal{F}_1))\times\Phi_2}$ *and* $G_{\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\neg f]\!]}(\mathcal{F}_1))\times\Phi_2}$ *are shown in Fig. 16 and Fig. 17 respectively, where the initial nodes are colored by* T*.* □

5. Generalized abstract models

We now define generalized MTSs which can be used as abstract models of FTSs that preserve CTL. They allow better precision of the abstract models, so that the validation or refutation of more CTL formulae can be established.

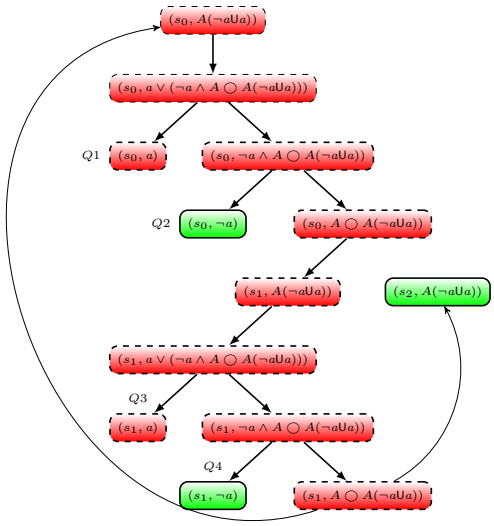

Figure 14: $G_{\pi_{[\![c \wedge \neg f]\!]}(\mathcal{F}_1) \times \Phi_1}$

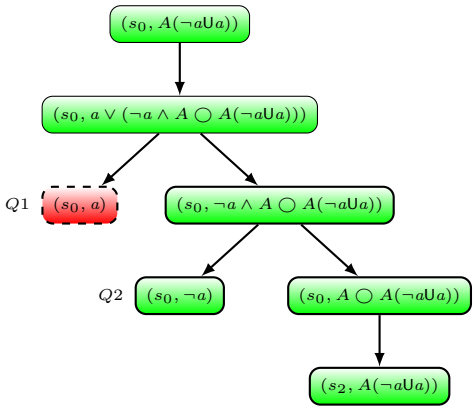

Figure 15: $G_{\pi_{[\![c \wedge f]\!]}(\mathcal{F}_1) \times \Phi_1}$

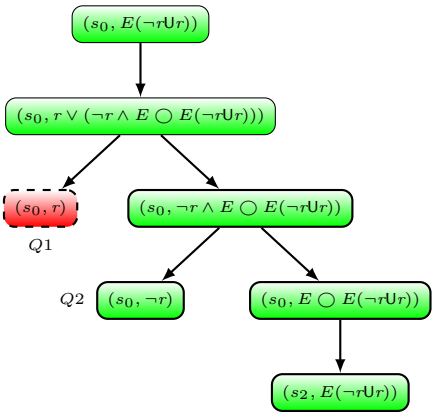

Figure 16: $G_{\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![f]\!]}(\mathcal{F}_1)) \times \Phi_2}$.

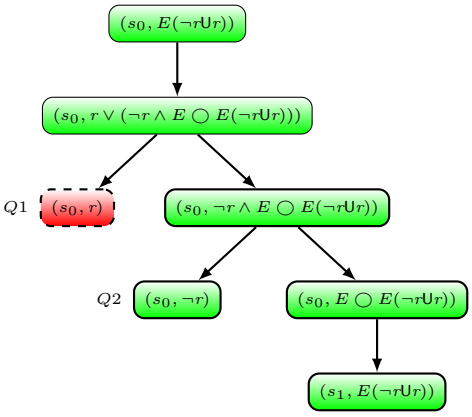

Figure 17: $G_{\boldsymbol{\alpha}^{\text{join}}(\pi_{[\![\neg f]\!]}(\mathcal{F}_1)) \times \Phi_2}$.

5.1. Generalized MTSs

The reason why the regular abstract model $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ in Fig. 3 does not satisfy $\Phi_2 = E(\neg r \mathsf{U} r)$ (see the evaluation in Fig. 5) comes from the current definition of must-transitions which is too restrictive, so that both $s_0 \xrightarrow{\neg f} s_1$ and $s_0 \xrightarrow{f} s_2$ in $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ are may-transitions. Hence, although both may-children $(s_1, E(\neg r \mathsf{U} r))$ and $(s_2, E(\neg r \mathsf{U} r))$ are colored by T, their parent $E\bigcirc$-node is colored by ?. Our goal is to suggest an alternative generalized definition of must-transitions that will weaken the previous (regular) definition. Inspired by [16, 17], we suggest the use of *hyper-transitions* to describe must-transitions.

Definition 17. *Given a set of states S, a* hyper-transition *is a pair (s, A), denoted $s \longrightarrow A$, where $s \in S$ and $A \subseteq S$ is a non-empty set of states.*

Given an FTS $\mathcal{F}$ with the set of configurations $\mathbb{K}$, recall that $t = s \longrightarrow s'$ is a must-transition in the regular abstract model $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})$ only if the transition t is present in all concrete variants $\pi_k(\mathcal{F})$ for $k \in \mathbb{K}$, that is $k \models \delta(t)$ for all $k \in \mathbb{K}$ (or, $\widetilde{\boldsymbol{\alpha}^{\text{join}}}(\delta(t)) = \mathit{true}$). The generalization aims to allow such a transition to exist from s to some state from A in all concrete variants $\pi_k(\mathcal{F})$ for $k \in \mathbb{K}$. This is achieved by using a *must hyper-transition* $s \longrightarrow A$.

Definition 18. *A* generalized modal transition system *(GMTS) is a tuple $\mathcal{M}^{gen} = (S, I, trans^{may}, trans^{must}, AP, L)$, where $trans^{may} \subseteq S \times S$ and $trans^{must} \subseteq S \times 2^S$, such that for every $(s, A) \in trans^{must}$ and $s' \in A$, we have $(s, s') \in trans^{may}$.*

An MTS can be seen as a GMTS where every must hyper-transition is a regular must-transition $s \longrightarrow \{s'\}$, so its target state is a singleton.

As before, a may-path in $\mathcal{M}^{gen}$ is an infinite path in $\mathcal{M}^{gen}$. However, instead of a must-path we now have a must hyper-path. Let Π be a set of paths, then $pref_i(\Pi)$ denotes the set of all prefixes of length i of paths in Π.

Definition 19. *A* must hyper-path *from a state s is a non-empty set Π of paths from s, such that for every $i \geq 0$:*

$$pref_{i+1}(\Pi) = \bigcup_{\rho_i \in pref_i(\Pi)} \{\rho_i \cdot s' \mid s' \in A_{\rho_i}\}$$

where for $\rho_i = s \cdot s_1 \ldots \cdot s_i \in pref_i(\Pi)$, the set $A_{\rho_i} \in 2^s$ is either such that $(s_i, A_{\rho_i}) \in trans^{must}$ or empty set if there is no must hyper-transition exiting s_i. Hence, a must hyper-path can include finite paths since A_{ρ_i} can be empty.

We use $[\![\mathcal{M}^{gen}]\!]^{\text{may}}_{GMTS}$ (resp., $[\![\mathcal{M}^{gen}]\!]^{\text{must}}_{GMTS}$) to denote the set of all may-paths (resp., must hyper-paths) in $\mathcal{M}^{gen}$ starting in an initial state.

We generalize the 3-valued CTL semantics for GMTSs. The semantics is defined similarly to the regular 3-valued CTL semantics for MTSs, except that the use of must-paths is replaced by must hyper-paths. More specifically, for a path formula ϕ and a must hyper-path Π, we define:

- $[\mathcal{M}^{gen}, \Pi \models^3 \phi] = tt\,(ff)$ iff for every $\rho \in \Pi$, we have that $[(\mathcal{M}^{gen}, \rho) \models^3 \phi] = tt\,(ff)$.
- otherwise, $[\mathcal{M}^{gen}, \Pi \models^3 \phi] = \bot$.

We now describe how to construct abstract GMTSs. Informally, a must hyper-transition $s \to A$ exists if in all valid variants $\pi_k(\mathcal{F})$ for $k \in \mathbb{K}$ there is a transition from s to some state s' from A.

Definition 20. *Given the FTS $\mathcal{F} = (S, I, trans, AP, L, \mathbb{F}, \mathbb{K}, \delta)$, we define its* abstraction *to be the GMTS $\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}) = (S, I, trans^{may}, trans^{must}, AP, L)$, where $trans^{may} = \{t \in trans \mid \boldsymbol{\alpha}^{\text{join}}(\delta(t)) = true\}$, and we have that $trans^{must} = \{(s, A) \mid \widetilde{\boldsymbol{\alpha}^{\text{join}}}(\vee_{s' \in A, t=(s,s') \in trans} \delta(t)) = true\}$.*

The preservation of full CTL is generalized from MTSs (see Theorem 5) to GMTSs as well.

Lemma 21. *Let $\Pi \in [\![\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F})]\!]^{must}_{GMTS}$. Then, for all $k \in \mathbb{K}$, there exists $\rho \in \Pi$, such that $\rho \in [\![\pi_k(\mathcal{F})]\!]_{TS}$.*

Proof. *Let $\Pi \in [\![\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F})]\!]^{must}_{GMTS}$. We claim:*

$$\forall i \geq 0, \forall k \in \mathbb{K}, \exists \rho_i \in pref_i(\Pi),\ s.t.\ \rho_i \in pref_i([\![\pi_k(\mathcal{F})]\!]_{TS}) \quad (*)$$

We prove the claim (*) by mathematical induction. For $i = 0$, (*) holds trivially. Assume that (*) holds for i. For $i + 1$, we have $pref_{i+1}(\Pi) = \bigcup_{\rho_i \in pref_i(\Pi)} \{\rho_i \cdot s' \mid s' \in A_{\rho_i}\}$, where $\rho_i = s_0 \cdot s_1 \cdot \ldots \cdot s_i$ and $(s_i, A_{\rho_i}) \in trans^{must}$. From IH, for all $k \in \mathbb{K}$, there exists $\rho_i \in pref_i(\Pi)$, s.t. $\rho_i \in pref_i([\![\pi_k(\mathcal{F})]\!]_{TS})$. Since $(s_i, A_{\rho_i}) \in trans^{must}$, we have $\widetilde{\boldsymbol{\alpha}^{\text{join}}}(\vee_{s' \in A_{\rho_i}} \delta((s_i, s'))) = true$. This means that for all $k \in \mathbb{K}$, there exists $s' \in A_{\rho_i}$ s.t. $k \models \delta(s_i, s')$, and so $\rho_i \cdot s' \in pref_{i+1}([\![\pi_k(\mathcal{F})]\!]_{TS})$. Thus, we conclude that (*) holds for $i + 1$. The proof follows from (*), when we consider all maximal paths in Π. □

Theorem 22 (Preservation results)**.** *For every $\Phi \in CTL$, we have:*

(1) $[\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}) \models^3 \Phi] = tt \implies [\mathcal{F} \models \Phi] = tt$ *and* $\forall k \in \mathbb{K}.[\pi_k(\mathcal{F}) \models \Phi] = tt.$

(2) $[\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}) \models^3 \Phi] = ff \implies [\mathcal{F} \models \Phi] = ff$ *and* $\forall k \in \mathbb{K}.[\pi_k(\mathcal{F}) \models \Phi] = ff.$

Proof. *By induction on the structure of* Φ*. All cases except* A *and* E *quantifiers are straightforward.*

*Consider the case (**1**):* $[\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}) \models^3 \Phi] = tt \implies [\mathcal{F} \models \Phi] = tt.$

Case $\Phi = A\phi$*. The proof is analogous to the respective case in Theor. 5.*

Case $\Phi = E\phi$*. To prove (**1**), we assume* $[\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}) \models^3 E\phi] = tt$*. This means that there exists a must hyper-path* $\Pi \in [\![\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F})]\!]^{must}_{GMTS}$ *such that* $[\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}), \Pi \models^3 \phi] = tt$*. By Lemma 21, we have that for all* $k \in \mathbb{K}$*, there exists* $\rho \in \Pi$ *s.t.* $\rho \in [\![\pi_k(\mathcal{F})]\!]_{TS}$*. Therefore,* $[\pi_k(\mathcal{F}) \models E\phi] = tt$ *for all* $k \in \mathbb{K}$*, and so* $[\mathcal{F} \models E\phi] = tt$*.*

*Consider the case (**2**):* $[\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}) \models^3 \Phi] = ff \implies [\mathcal{F} \models \Phi] = ff.$

Case $\Phi = A\phi$*. To prove (**2**), we assume* $[\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}) \models^3 A\phi] = ff$*. This means that there exists a must hyper-path* $\Pi \in [\![\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F})]\!]^{must}_{GMTS}$ *such that* $[\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}), \Pi \models^3 \phi] = ff$*. By Lemma 21, we have that for all* $k \in \mathbb{K}$*, there exists* $\rho \in \Pi$ *s.t.* $\rho \in [\![\pi_k(\mathcal{F})]\!]_{TS}$*. Therefore,* $[\pi_k(\mathcal{F}) \models A\phi] = ff$ *for all* $k \in \mathbb{K}$*, and so* $[\mathcal{F} \models A\phi] = ff$*.*

Case $\Phi = E\phi$*. The proof is analogous to the respective case in Theor. 5.* □

The use of GMTSs allows construction of abstract models $\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F})$ that are more precise than regular abstract models $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})$ described as MTSs. This is shown by making more efficient the abstraction-refinement procedure for verifying $[\mathcal{F}_1 \models \Phi_2]$ in Examples 11, 15, 16.

Example 23. *Given the FTS* $\mathcal{F}_1$ *in Fig. 1, the constructed generalized abstract model* $\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}_1)$ *is shown in Fig. 19. There exists one must hyper-transition* $s_0 \longrightarrow \{s_1, s_2\}$ *in* $\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}_1)$*. We can see that* $[\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}_1) \models^3 \Phi_2] = tt$*, since there exists a must hyper-path* $\{s_0 s_1 \ldots, s_0 s_2 \ldots\}$ *that satisfies* $\Phi_2 = E(\neg r \mathsf{U} r)$*. Therefore, we conclude that all variants of* $\mathcal{F}_1$ *satisfy* Φ_2 *in the first iteration, so there is no need to refine the initial abstract model. In contrast, note that* $s_0 \longrightarrow s_1$ *and* $s_0 \longrightarrow s_2$ *are may-transitions in* $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$*, and so* $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1) \models^3 \Phi_2] = \bot$ *as shown in Example 11.* □

Any abstract GMTS can be reduced without damaging its precision, based on the following observation. Given two must hyper-transitions $s \longrightarrow A$ and $s \longrightarrow A'$, where $A \subseteq A'$, the transition $s \longrightarrow A'$ can be discarded without sacrificing the precision of the GMTS. Therefore, a possible optimization would be to use only *minimal* hyper-transitions where A' is minimal.

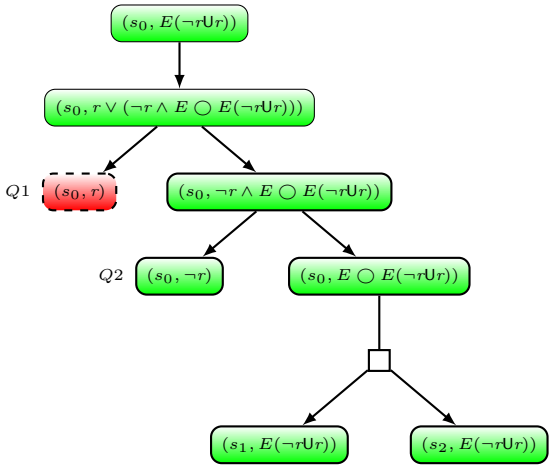

Figure 18: $G_{\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}_1) \times \Phi_2}$.

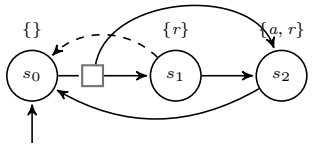

Figure 19: GMTS $\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}_1)$.

6. Generalized abstraction-refinement framework

We show how GMTSs can be used in practice within an abstraction-refinement framework. Given a concrete FTS $\mathcal{F}$ with the set of configurations $\mathbb{K}$, we compute the abstract GMTS $\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F})$ as follows:

- We first construct an abstract model $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})$, which includes its may-transitions and its regular must-transitions.
- For every state, add a must hyper-transition targeting the set of all target states of its outgoing may-transitions such that the disjunction of all feature expressions guarding those transitions in $\mathcal{F}$ is $\mathbb{K}$.
- discard any must hyper-transition $s \longrightarrow A$ that is not minimal, which means that there is $s \longrightarrow A'$ where $A' \subseteq A$.

We now complete the generalized abstraction-refinement framework by providing a model checking algorithm that evaluates CTL formulae over GMTSs and a suitable refinement mechanism.

As a model checking algorithm we suggest a simple generalization of the game-based algorithm presented in Section 3. The only change is in the definition of must-children as well as in the part of the coloring algorithm that depends on must-children. *Must hyper-edges* are edges based on must hyper-transitions of GMTSs. Nodes of a set A are a *must hyper-child* of the node n if there exists a must hyper-edge (n, A). In the coloring algorithm, we add the following cases:

- An $A\bigcirc$ node is colored by F if it has a must hyper-child colored by F.
- An $E\bigcirc$ node is colored by T if it has a must hyper-child colored by T.

As for the refinement mechanism, we can use Algorithm 1 suggested in Section 4 in order to find a failure state, analyze the failure, and decide how to split the abstract configuration. This is due to the fact that the refinement is based on may-transitions only. Therefore, no additional change is needed.

Theorem 24. *The generalized abstraction-refinement procedure, denoted as* $\texttt{Gen-Verify}(\mathcal{F}, \mathbb{K}, \Phi)$*, is guaranteed to terminate and is correct.*

Proof. *The proof for termination is analogous to the corresponding proof for* $\texttt{Verify}(\mathcal{F}, \mathbb{K}, \Phi)$ *(see Theorem 14). This is due to the fact that failure reasons are may-transitions, which are treated in the same way in both cases.*

The proof for correctness follows from Theorem 22, so that case (1) *handles the case when* $\texttt{Gen-Verify}(\mathcal{F}, \mathbb{K}, \Phi)$ *returns answer "tt", while case* (2) *when* $\texttt{Gen-Verify}(\mathcal{F}, \mathbb{K}, \Phi)$ *returns answer "ff".* □

Example 25. *We now show how to evaluate* $[\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}_1) \models^3 \Phi_2]$ *using the generalized game-based model checking algorithm. The graph* $G_{\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}_1)\times\Phi_2}$ *is shown in Fig. 18. In this case, the node* $(s_0, E \bigcirc E(\neg r \mathsf{U} r))$ *has a must hyper-child* $\{(s_1, E(\neg r \mathsf{U} r)), (s_2, E(\neg r \mathsf{U} r))\}$ *which is colored by* T*. Hence,* $(s_0, E \bigcirc E(\neg r \mathsf{U} r))$ *will be colored by* T*, which makes the initial node colored by* T *as well. Thus, we obtain that* $[\boldsymbol{\alpha}^{\text{gen-join}}(\mathcal{F}_1) \models^3 \Phi_2] = tt$*. Note that in Fig. 18, for readability we do not draw the subgraphs of* $(s_1, E(\neg r \mathsf{U} r))$ *and* $(s_2, E(\neg r \mathsf{U} r))$*, but they are the same as in Fig. 5.*

To conclude, $\texttt{Gen-Verify}(\mathcal{F}_1, \mathbb{K}_1, \Phi_2)$ *needs one iteration and one colored game-graph to evaluate* $[\mathcal{F}_1 \models \Phi_2]$*, whereas* $\texttt{Verify}(\mathcal{F}_1, \mathbb{K}_1, \Phi_2)$ *needs two iterations and three colored game-graphs to evaluate* $[\mathcal{F}_1 \models \Phi_2]$*.* □

7. Evaluation

We evaluate our abstraction-refinement procedures for verifying CTL properties of reactive system families. They consist of generating initial abstract models, then verifying CTL properties on them using the game-based model checking algorithm. In case of indefinite results, refined abstract models are generated and the whole procedure is repeated on them. The evaluation aims to show that we can efficiently verify some interesting CTL properties of different system families using our abstraction-refinement approaches.

7.1. Experimental setup

To evaluate our approach, we consider two case studies and a dozen of CTL properties. We use a synthetic example to demonstrate specific characteristics of our approach, and the ELEVATOR model which is often used as benchmark in the SPL community [23, 24, 18, 9].

We have implemented the game-based model checking algorithm as well as its generalized extension in Java. They are used as basis to implement our abstraction-refinement procedures. We compare (1) our abstraction-refinement procedure `Verify` vs. (2) our generalized abstraction-refinement procedure `Gen-Verify` vs. (3) the plain lifted model checking algorithm implemented by the lifted (extended) version of NUSMV model checker, denoted fNUSMV [18]. fNUSMV uses symbolic algorithms to model check variational systems over CTL properties at once. It uses a feature-oriented extension of the NUSMV language [23] as input, which is shown to be a high-level representation of FTSs [18]. In contrast to `Verify` and `Gen-Verify`, fNUSMV uses no abstractions.

The BDD model checker NUSMV (version 2.5.4) is run with the parameter `-df -dynamic`, which ensures that the BDD package reorders the variables during verification in case the BDD size grows beyond a certain threshold. The reported performance numbers constitute the average runtime of five independent executions. For each experiment, we measure TIME to perform an analysis task, and CALL which is the number of times an approach calls the model checking engine. We say that a task is *infeasible* when it is taking more time than the given timeout threshold, which we set on two hours.

All experiments were executed on a 64-bit Intel®Core^{TM} i5-3337U CPU running at 1.80 GHz with 8 GB memory. The implementation, benchmarks, and all results obtained from our experiments are available from: `https://aleksdimovski.github.io/automatic-ctl.html`.

7.2. Synthetic example

First, we have tested the limits of the lifted model checking as implemented in fNUSMV. Although fNUSMV symbolically analyzes variability models in a single run, it still greatly depends on the size of configuration space $|\mathbb{K}|$. Combinatorially, $|\mathbb{K}|$ grows exponentially with the number of features $|\mathbb{F}|$. Consequently, for variability models with large $|\mathbb{F}|$, analysis may become impractically slow or infeasible. In those cases, we can use abstraction-refinement procedures, `Verify` and `Gen-Verify`, to reduce the configuration

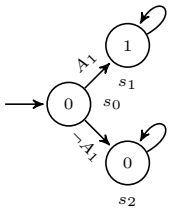

Figure 20: M_1.

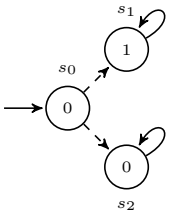

Figure 21: $\boldsymbol{\alpha}^{\text{join}}(M_1)$.

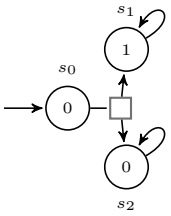

Figure 22: $\boldsymbol{\alpha}^{\text{gen-join}}(M_1)$.

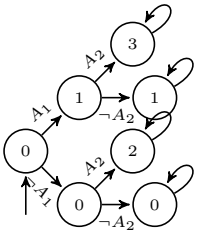

Figure 23: FTS M_2.

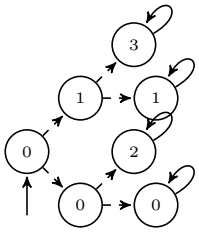

Figure 24: MTS $\boldsymbol{\alpha}^{\text{join}}(M_2)$.

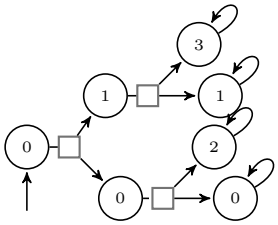

Figure 25: GMTS $\boldsymbol{\alpha}^{\text{gen-join}}(M_2)$

space and thus obtain feasible lifted verification of such highly-configurable variability models.

In order to confirm the above statement, we have constructed FTSs M_n (where $n > 0$), which consist of n features $A_1, \ldots, A_n$ and an integer data variable x, such that the set AP consists of all evaluations of x which assign integer values to x. The set of valid configurations is $\mathbb{K}_n = 2^{\{A_1,\ldots,A_n\}}$. The FTS M_n has a tree-like structure. The root is the initial state with $x = 0$. Each level k ($k \geq 1$) contains two states reachable with two transitions leading from a state from a previous level. One transition is allowable for variants with feature A_k enabled, so that in the target state the variable's value is $x + 2^{k-1}$ where x is its value in the source state, whereas the other transition is allowable for variants with feature A_k disabled, so that the values of x are the same in the source and target states. For example, FTSs M_1 and M_2 are shown in Fig. 20 and Fig. 23 respectively, where in each state we show the current value of x. The abstract models $\boldsymbol{\alpha}^{\text{join}}(M_1)$ and $\boldsymbol{\alpha}^{\text{join}}(M_2)$ are given in Fig. 21 and Fig. 24, while the generalized abstract models $\boldsymbol{\alpha}^{\text{gen-join}}(M_1)$ and $\boldsymbol{\alpha}^{\text{gen-join}}(M_2)$ are given in Fig. 22 and Fig. 25. Note that there is a must hyper-transition from a state to the two states in the next level since one of those two states can be reached in all valid variants.

We consider four properties: $\Phi_1 = A(\mathit{true}\,\mathsf{U}(x \geq 0))$, $\Phi_2 = A(\mathit{true}\,\mathsf{U}(x \geq 1))$, $\Phi_3 = E \bigcirc E(\mathit{true}\,\mathsf{U}(x \geq 1))$, and $\Phi_4 = A(\mathit{true}\,\mathsf{U}(x < 0))$. We have verified M_n

n	Φ_1			Φ_2			Φ_3			Φ_4		
	fNuSMV	`Verify`	`Gen-Ver`	fNuSMV	`Verify`	`Gen-Ver.`	fNuSMV	`Verify`	`Gen-Ver`	fNuSMV	`Verify`	`Gen-Ver`
2	0.04	0.47	0.51	0.03	1.18	1.21	0.03	0.67	0.48	0.03	1.70	0.73
7	0.08	0.97	0.97	0.10	5.20	5.39	0.08	1.22	1.00	0.09	124.92	8.84
10	141.4	3.76	3.90	150.0	11.51	11.46	297.1	4.11	3.87	316.3	932.2	66.7
11	timeout	6.89	6.90	timeout	16.11	16.22	timeout	7.28	7.05	timeout	timeout	133.1
15	timeout	103.2	103.9	timeout	124.5	125.1	timeout	101.9	99.6	timeout	timeout	2511

Figure 26: Verifying M_n (sec) using lifted fNuSMVvs. `Verify` vs. `Gen-Verify`.

against Φ_1, Φ_2, Φ_3, and Φ_4 using fNuSMV(e.g. see fNuSMVmodels for M_1 and M_2 in [25, Appendix E]), as well as using our `Verify` and `Gen-Verify`.

The property $\Phi_1 = A(true\,\mathsf{U}(x \geq 0))$ is satisfied by all variants in $\mathbb{K}_n$. `Verify` and `Gen-Verify` terminate in one iteration since $\boldsymbol{\alpha}^{\text{join}}(M_n)$ satisfies Φ_1. See the colored game-graph $G_{\boldsymbol{\alpha}^{\text{join}}(M_1)\times\Phi_1}$ in Fig. 27.

The property $\Phi_2 = A(true\,\mathsf{U}(x \geq 1))$ is violated only by one configuration $\neg A_1 \wedge \ldots \wedge \neg A_n$ (where all features are disabled, so there is a single path in that variant where all states have $x = 0$). `Verify` and `Gen-Verify` need $n + 1$ iterations. First, an indefinite result is reported for $\boldsymbol{\alpha}^{\text{join}}(M_n)$, e.g. see $G_{\boldsymbol{\alpha}^{\text{join}}(M_1)\times\Phi_2}$ in Fig. 28, and the configuration space is split into $[\![\neg A_1]\!]$ and $[\![A_1]\!]$ subsets. For the latter, $\pi_{[\![A_1]\!]}(M_n)$, we obtain an affirmative answer, whereas for the former, $\pi_{[\![\neg A_1]\!]}(M_n)$, we obtain an indefinite result with the refinement based on $[\![\neg A_2]\!]$. The refinement procedure proceeds in this way until we obtain definite results for all variants.

The property $\Phi_3 = E\bigcirc E(true\,\mathsf{U}(x \geq 0))$ is satisfied by all variants in $\mathbb{K}_n$. `Verify` needs 2 iterations to terminate, such that in the first iteration the result is ? and the configuration space is split into $[\![A_1]\!]$ and $[\![\neg A_1]\!]$. This is due to the fact that an $E\bigcirc$-node is colored ? if it has a may-child colored T. `Gen-Verify` needs only one iteration to terminate since $\boldsymbol{\alpha}^{\text{gen-join}}(M_n)$ satisfies Φ_4. This is due to the fact that an $E\bigcirc$-node is colored T if it has a must hyper-child colored T. Therefore, `Gen-Verify` will always slightly outperform `Verify` on Φ_3.

The property $\Phi_4 = A(true\,\mathsf{U}(x < 0))$ is violated by all variants in $\mathbb{K}_n$. `Verify` needs n iterations to terminate, such that in the first iteration the configuration space is split into $[\![A_n]\!]$ and $[\![\neg A_n]\!]$, in the second iteration into $[\![A_{n-1} \wedge A_n]\!]$, $[\![A_{n-1} \wedge \neg A_n]\!]$, $[\![\neg A_{n-1} \wedge A_n]\!]$ and $[\![\neg A_{n-1} \wedge \neg A_n]\!]$, and so on until the $(n + 1)$-th iteration when we consider in the brute-force fashion all

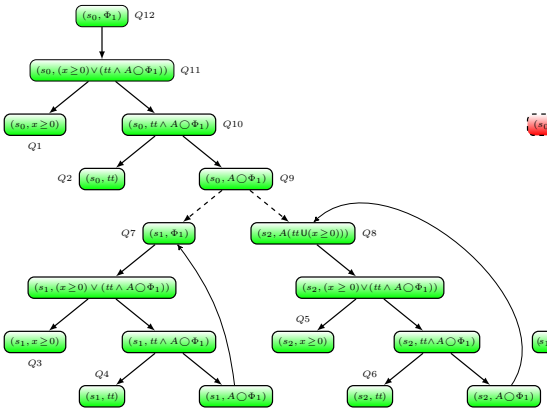

Figure 27: $G_{\boldsymbol{\alpha}^{\text{join}}(M_1)\times\Phi_1}$.

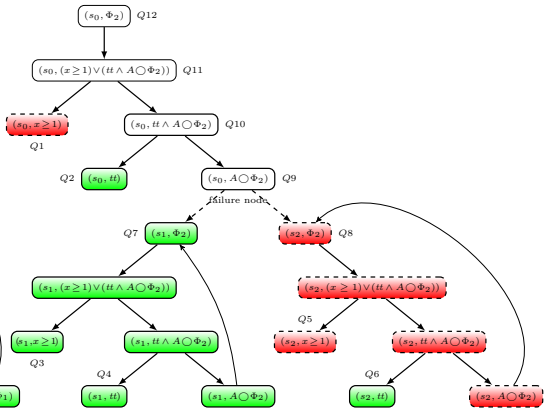

Figure 28: $G_{\boldsymbol{\alpha}^{\text{join}}(M_1)\times\Phi_2}$.

variants. For example, see the colored game-graph $G_{\boldsymbol{\alpha}^{\text{join}}(M_1)\times\Phi_4}$ in Fig. 29. The initial node is colored ? due to the fact that an $A\bigcirc$-node is colored ? if it has a may-child colored F. On the other hand, `Gen-Verify` terminates in only one iteration since $\boldsymbol{\alpha}^{\text{gen-join}}(M_n)$ satisfies Φ_4. The colored game-graph $G_{\boldsymbol{\alpha}^{\text{gen-join}}(M_1)\times\Phi_4}$ is shown in Fig. 30. Note that, in this case an $A\bigcirc$-node will be colored F since its must hyper-child is colored by F. Hence, Φ_4 represents the worst case for `Verify` and the best case for `Gen-Verify`.

The performance results are shown in Fig. 26. Notice that fNuSMV reports all results in only one iteration. Yet, for $n = 11$ (for which $|\mathbb{K}| = 2^{11}$), it timeouts after 2 hours. The state space of models M_n generated by fNuSMV grows exponentially with the number of features, $n = |\mathbb{F}|$. Thus, for M_n with larger n, verification tasks quickly become very prohibitive. On the other hand, `Verify` and `Gen-Verify` are feasible and very efficient even for large values of n. `Verify` and `Gen-Verify` run within the same time for Φ_1 and Φ_2. However, `Gen-Verify` is more efficient than `Verify` for Φ_3 and Φ_4, since it terminates in only one iteration for them. This is especially visible for Φ_4, which represents the worst case for `Verify`. In this case, all variants are verified in a brute force fashion plus the overhead for refinements.

7.3. Elevator

We have experimented with the Elevator model with four floors, designed by Plath and Ryan [23]. The model contains about 300 LOC of fNuSMV code and 9 independent optional features that modify the basic behaviour of the elevator, thus yielding $2^9 = 512$ variants. The features

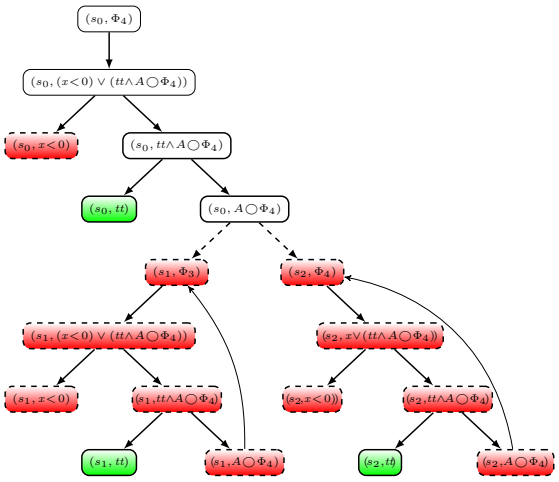

Figure 29: $G_{\boldsymbol{\alpha}^{\text{join}}(M_1) \times \Phi_4}$.

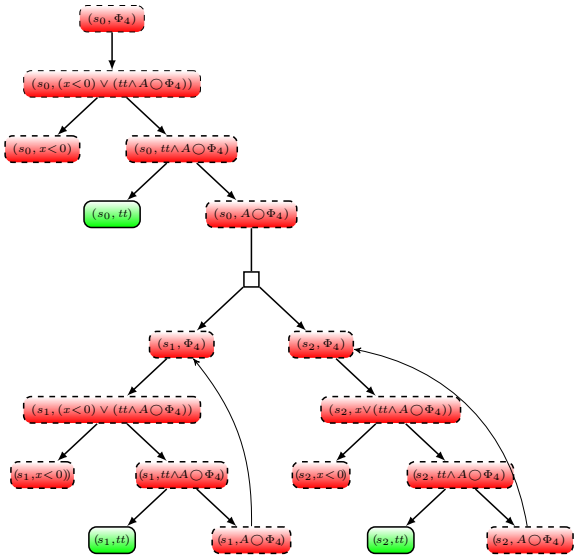

Figure 30: $G_{\boldsymbol{\alpha}^{\text{gen-join}}(M_1) \times \Phi_4}$.

are: `Antiprunk`, `Empty`, `Exec`, `OpenIfIdle`, `Overload`, `Park`, `QuickClose`, `Shuttle`, and `TTFull`. To use our `Verify` and `Gen-Verify` procedures, we have manually translated the fNuSMV model into an FTS and then we have called `Verify` and `Gen-Verify` on it. The basic Elevator system consists of a single lift that travels between four floors. There are four platform buttons and a single lift, which declares variables $floor, door, direction$, and a further four cabin buttons. The lift will always serve all requests in its current direction before it stops and changes direction. When serving a floor, the lift door opens and closes again.

We consider four properties. The property "$\Phi_1 = E(tt \mathsf{U}(floor = 1 \wedge idle \wedge door = closed))$" states that there exists a path containing a state where the lift is on the first floor, idle, and the door is closed, whereas "$\Phi_2 = A(tt \mathsf{U}(floor = 1 \wedge idle \wedge door = closed))$" claims that the above state exists on all possible paths. The property "$\Phi_3 = E(tt \mathsf{U}((floor = 3 \wedge \neg liftBut3.pressed \wedge direction = up) \implies door = closed))$" is that, there exists a path which contains a state such that if the lift is on the third floor, the lift button is pressed and the direction is up, then the lift door is closed. The property "$\Phi_4 = E(tt \mathsf{U}(A \bigcirc (door = closed))$" is that, there exists a path on which eventually there is a state such that in all its next states the lift door is closed. The performance results are shown in Fig. 31. The properties

prop.	fNuSMV	`Verify`	`Gen-Verify`	*Improvement*
Φ_1	14.28 s	1.64 s	1.75 s	9 ×
Φ_2	1.59 s	1.07 s	1.12 s	1.5 ×
Φ_3	1.76 s	1.02 s	1.12 s	1.7 ×
Φ_4	1.82 s	1.22 s	1.29 s	1.5 ×

Figure 31: Verification of Elevator properties (Time in seconds). Improvement shows the speed-up of `Verify` and `Gen-Verify` vs. fNuSMV.

Φ_1 and Φ_2 are satisfied by all variants, thus `Verify` and `Gen-Verify` achieve speed-ups of 9 times for Φ_1 and 1.5 times for Φ_2 compared to the fNuSMV approach. fNuSMV takes 1.76 sec to check Φ_3 and 1.82 to check Φ_4, whereas `Verify` and `Gen-Verify` run in 1.02 seconds for Φ_3 and 1.22 for Φ_4, thus giving 1.7 times performance speed-up for Φ_3 and 1.5 times speed-up for Φ_4.

7.4. Discussion

In conclusion, the evaluation shows that for certain properties and variability models, our abstraction-refinement procedures, `Verify` and `Gen-Verify`, can outperform the plain lifted model checking fNuSMV. The abstraction-refinement procedures achieve the best results when the property to be checked is either satisfied by all variants (e.g., Φ_1 and Φ_2 for both examples) or there exists a subset of erroneous variants that share a common counter-example and depend on only few features. The worst case is when every variant triggers a different counter-example, so our abstraction-refinement procedures end up in verifying all variants one by one in a brute force fashion plus the overhead for generating and verifying all intermediate abstract models (e.g., Φ_4 for the synthetic example and `Verify`).

Note that NuSMV is a highly-optimized industrial-strength tool compared to our proof-of-concept implementations of `Verify` and `Gen-Verify`. NuSMV contains many optimisation algorithms, which are result of more than three decades research on advanced computer aided verification. On the other hand, `Verify` and `Gen-Verify` are research prototype tools developed by the first author to support algorithms proposed in this work. Still, for models with high variability (larger values of $|\mathbb{F}|$) and certain properties, our approach can be faster than fNuSMV.

8. Abstraction-refinement for modal μ-calculus

We now discuss how to extend our abstraction-refinement procedures by considering a richer set of temporal properties, as expressed in the modal μ-calculus [26].

8.1. Syntax and semantics

The modal μ-calculus logic [26], denoted L_μ, is defined as:

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

where $x \in \mathit{Var}$ ranges over propositional variables. Note that L_μ formulae φ are given in negation normal form. Intuitively, $\Box$ stands for "all successors", and $\Diamond$ stands for "exists a successor"; while μ denotes a least fixpoint, and ν denotes greatest fixpoint. We will also write η for either μ or ν. We assume that every variable x identifies a unique subformula $fp_\varphi(x) = \nu x.\varphi'$ of φ.

The semantics $[\![\varphi]\!]^{\mathcal{T}}_\rho$ of a L_μ formula φ over a TS $\mathcal{T}$ and an environment $\rho : \mathit{Var} \to 2^S$, which binds variables to sets of states where they hold, is:

(1) $[\![a]\!]^{\mathcal{T}}_\rho(s) = tt$ iff $a \in L(s)$; $\quad [\![\neg a]\!]^{\mathcal{T}}_\rho(s) = tt$ iff $a \notin L(s)$

(2) $[\![\varphi_1 \wedge \varphi_2]\!]^{\mathcal{T}}_\rho(s) = tt$ iff $[\![\varphi_1]\!]^{\mathcal{T}}_\rho(s) = tt$ and $[\![\varphi_2]\!]^{\mathcal{T}}_\rho(s) = tt$;
$[\![\varphi_1 \vee \varphi_2]\!]^{\mathcal{T}}_\rho(s) = tt$ iff $[\![\varphi_1]\!]^{\mathcal{T}}_\rho(s) = tt$ or $[\![\varphi_2]\!]^{\mathcal{T}}_\rho(s) = tt$

(3) $[\![\Box\varphi]\!]^{\mathcal{T}}_\rho(s) = tt$ iff $\forall s' \in S.(s, s') \in trans \implies [\![\varphi]\!]^{\mathcal{T}}_\rho(s') = tt$
$[\![\Diamond\varphi]\!]^{\mathcal{T}}_\rho(s) = tt$ iff $\exists s' \in S.(s, s') \in trans \wedge [\![\varphi]\!]^{\mathcal{T}}_\rho(s') = tt$

(4) $[\![\mu x.\varphi]\!]^{\mathcal{T}}_\rho(s) = \mathrm{lfp}(\lambda g.[\![\varphi]\!]^{\mathcal{T}}_{\rho[x \mapsto g]})$; $\quad [\![\nu x.\varphi]\!]^{\mathcal{T}}_\rho(s) = \mathrm{gfp}(\lambda g.[\![\varphi]\!]^{\mathcal{T}}_{\rho[x \mapsto g]})\}$

where $\rho[x \mapsto g]$ is the environment which is the same as ρ, except that x is mapped to g, and $\mathrm{lfp}(f)$, $\mathrm{gfp}(f)$ stand for the least and greatest fixpoints of functional f. For a closed formula φ, we write $[\mathcal{T}, s \models \varphi] = [\![\varphi]\!]^{\mathcal{T}}_{\perp_{env}}(s)$, where $\perp_{env}$ maps every $x \in \mathit{Var}$ to $\emptyset$. We write $[\mathcal{T} \models \varphi] = tt$, iff all its initial states satisfy the formula: $\forall s_0 \in I.\, [\mathcal{T}, s_0 \models \varphi] = tt$. We say that an FTS $\mathcal{F}$ *satisfies* a μ-calculus formula φ, written $[\mathcal{F} \models \varphi] = tt$, iff all its valid variants satisfy the formula: $\forall k \in \mathbb{K}.\, [\pi_k(\mathcal{F}) \models \varphi] = tt$.

The semantics of L_μ over an MTS $\mathcal{M}$ is slightly different from the semantics for TSs. We define the 3-valued semantics $[\![\varphi]\!]^{\mathcal{M}}_{\rho,3}$ of φ over an MTS $\mathcal{M}$. The semantics of modalities $\Box$ and $\Diamond$ is extended to the 3-valued case as follows:

(3) $[\![\Box\varphi]\!]^{\mathcal{M}}_{\rho,3}(s) = \begin{cases} tt, & \text{if } \forall s' \in S.\,(s,s') \in trans^{\text{may}} \implies [\![\varphi]\!]^{\mathcal{M}}_{\rho,3}(s') = tt \\ ff, & \text{if } \exists s' \in S.\,(s,s') \in trans^{\text{must}} \implies [\![\varphi]\!]^{\mathcal{M}}_{\rho,3}(s') = ff \\ \bot, & \text{otherwise} \end{cases}$

$[\![\Diamond\varphi]\!]^{\mathcal{M}}_{\rho,3}(s) = \begin{cases} tt, & \text{if } \exists s' \in S.\,(s,s') \in trans^{\text{must}} \implies [\![\varphi]\!]^{\mathcal{M}}_{\rho,3}(s') = tt \\ ff, & \text{if } \forall s' \in S.\,(s,s') \in trans^{\text{may}} \implies [\![\varphi]\!]^{\mathcal{M}}_{\rho,3}(s') = ff \\ \bot, & \text{otherwise} \end{cases}$

For a closed formula φ, we write $[\mathcal{M}, s \models^3 \varphi] = [\![\varphi]\!]^{\mathcal{T}}_{\bot_{env},3}(s)$. We write $[\mathcal{M} \models^3 \varphi] = tt$, iff $\forall s_0 \in I.\,[\mathcal{M}, s_0 \models^3 \Phi] = tt$, while $[\mathcal{M} \models^3 \varphi] = ff$, iff $\exists s_0 \in I.\,[\mathcal{M}, s_0 \models^3 \Phi] = ff$. We then show that the MTS $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})$ preserves the modal μ-calculus L_μ.

Theorem 26 (Preservation results)**.** *For every closed formula $\varphi \in L_\mu$,*

(1) $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 \varphi] = tt \implies [\mathcal{F} \models \varphi] = tt.$

(2) $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 \varphi] = ff \implies [\mathcal{F} \models \varphi] = ff.$

Proof. By induction on the structure of φ.
We consider the case (**1**) and $\varphi = \Box\varphi'$. We proceed by contraposition. Assume $[\mathcal{F} \models \Box\varphi'] \neq tt$. Then, there exists a configuration $k \in \mathbb{K}$ and a transition $(s_0, s_1) \in trans$ of $\pi_k(\mathcal{F})$ (where $s_0 \in I$ of $\pi_k(\mathcal{F})$), such that $[\pi_k(\mathcal{F}), s_1 \models \varphi'] \neq tt$. By definitions of $\boldsymbol{\alpha}^{\text{join}}$ and $trans^{\text{may}}$, we have $(s_0, s_1) \in trans^{\text{may}}$ of $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})$, and so $[\![\Box\varphi']\!]^{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F})}_{\bot_{env},3}(s_0) \neq tt$ and $[\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}) \models^3 \Box\varphi'] \neq tt$ by definition. □

8.2. Abstraction-refinement procedure

The 3-valued model checking game for μ-calculus L_μ over MTSs is defined in [27, 14]. The game is played by Player $\forall$ and Player $\exists$ in order to evaluate a L_μ-formula φ in a state s of $\mathcal{M}$. We now briefly describe the model checking game for μ-calculus [27, 14], and we emphasize the points in which it differs from the corresponding game for CTL in Section 3.

Configurations are elements of $S \times sub(\varphi)$. Some possible moves are:

(1) if $C_i = (s, \Box\varphi')$, Player $\forall$ chooses a must-transition $s \longrightarrow s'$ (for refutation) or a may-transition $s \longrightarrow s'$ of $\mathcal{M}$ (to prevent satisfaction), and $C_{i+1} = (s', \varphi')$.

(3) if $C_i = (s, \Diamond\varphi')$, Player $\exists$ chooses a must-transition $s \longrightarrow s'$ (for satisfaction) or a may-transition $s \longrightarrow s'$ of $\mathcal{M}$ (to prevent refutation), and $C_{i+1} = (s', \varphi')$.

(3) if $C_i = (s, \eta x.\varphi')$, then $C_{i+1} = (s, x)$.

(4) if $C_i = (s, x)$, then $C_{i+1} = (s, \varphi')$ where $fp_\varphi(x) = \nu x.\varphi'$.

The moves $(3) - (4)$ are deterministic, thus any player can make them. The moves for literals, $\wedge$, and $\vee$ are the same as for CTL games (see Section 3).

The game-graph $G_{\mathcal{M}\times\varphi}$ of the 3-valued model checking game contains all the information relevant for the model checking. The set of nodes N is a subset of configurations $S \times sub(\varphi)$, where (s_0, φ) with $s_0 \in I$ is the initial node. The rest of nodes and edges are defined by possible moves at each node (configuration). We also define a priority function $\Theta_{\mathcal{M}\times\varphi} : Var \to \mathbb{N}$ that maps each variable to a priority [27, 14]. Let $x_1, \ldots, x_n$ be all variables in φ. Then, $\Theta_{\mathcal{M}\times\varphi}(x_i)$ is even iff x_i is of type ν, and $\Theta_{\mathcal{M}\times\varphi}(x_i)$ is odd iff x_i is of type μ. Also, $\Theta_{\mathcal{M}\times\varphi}(x_i) \leq \Theta_{\mathcal{M}\times\varphi}(x_j)$ whenever x_j occurs freely in $fp_\phi(x_i)$.

The coloring algorithm [27, 14] is performed by solving the 3-valued parity game $(G_{\mathcal{M}\times\varphi}, \Theta_{\mathcal{M}\times\varphi})$, where each color T, F, ? stands for a possible result (winner) in the game. It represents a generalization of Zielonka's algorithm for solving 2-valued parity games. The result of the coloring algorithm is a 3-valued coloring function $\chi : N \to \{T, F, ?\}$, which reflects the 3-valued semantics of μ-calculus.

Theorem 27 ([27, 14])**.** *Let $\mathcal{M}$ be an MTS and φ be a μ-calculus formula. For each $n = (s, \varphi') \in G_{\mathcal{M}\times\varphi}$:*

(1) $[\mathcal{M}, s \models^3 \varphi'] = tt$ *iff* $\chi(n) = T$ *iff Player* $\exists$ *has a winning strategy at* n.

(2) $[\mathcal{M}, s \models^3 \varphi'] = ff$ *iff* $\chi(n) = F$ *iff Player* $\forall$ *has a winning strategy at* n.

(3) $[\mathcal{M}, s \models^3 \varphi'] = \bot$ *iff* $\chi(n) = ?$ *iff none of the players has a winning strategy at* n.

Example 28. *Figure 32 presents the game-graph for the MTS $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ from Fig. 3 and the L_μ formula $\varphi_1 = \mu x.(a \vee (\neg a \wedge \Box x))$, which is equivalent to the CTL formula $\Phi_1 = A(\neg a U a)$. The priority function assigns priority 1 for nodes (s_0, x), (s_1, x), and (s_2, x), since the fixpoint formula of x in φ_1 is of type μ. The initial node (s_0, φ_1) is colored by ?, which reflects the fact that the value of φ_1 in the initial state s_0 of $\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)$ is $\bot$.* □

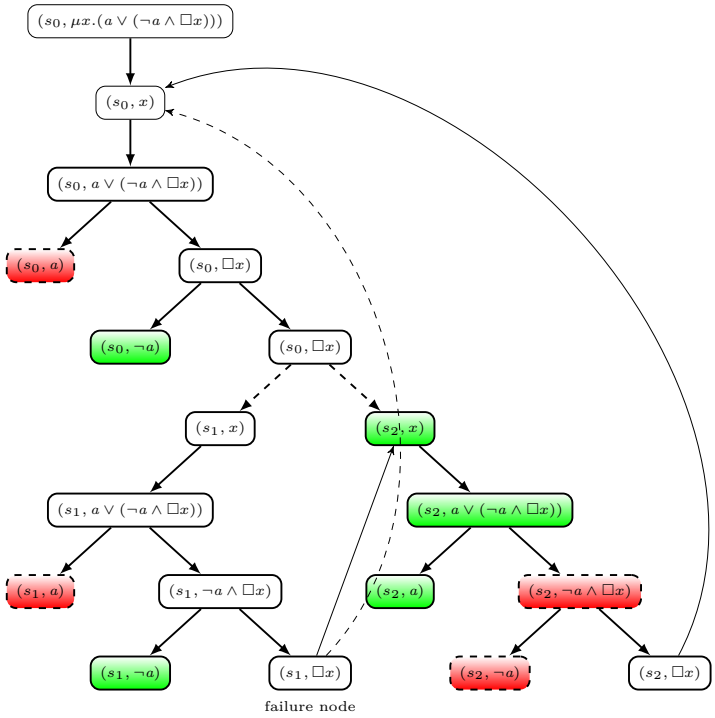

Figure 32: The colored $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)\times\varphi_1}$.

If the model checking result of an abstract model is indefinite ($\perp$), refinement is needed. The coloring algorithm [27, 14] reports a failure state and a failure reason in case of an indefinite result. The failure reason is an outgoing may-transition of the failure state in the underlying abstract model that is not a must-transition. In the same way as for CTL games in Section 4, refinement is then performed by splitting abstract configurations in a way that eliminates the failure reason.

Example 29. *The failure analysis on the game-graph $G_{\boldsymbol{\alpha}^{\text{join}}(\mathcal{F}_1)\times\varphi_1}$ in Fig. 32 reports that the failure node is $(s_1, \Box x)$ and the failure reason is the may-edge $(s_1, \Box x) \longrightarrow (s_0, \Box x)$. The corresponding concrete transition in $\mathcal{F}_1$ is $s_1 \xrightarrow{c} s_0$. Hence, we partition the configuration set $\mathbb{K}_1$ of $\mathcal{F}_1$ into subsets $[\![c]\!]$ and $[\![\neg c]\!]$, and in the next iteration we consider FTSs $\pi_{[\![c]\!]}(\mathcal{F}_1)$ and $\pi_{[\![\neg c]\!]}(\mathcal{F}_1)$.* □

9. Related work

Lifted model checking has been a subject of active research in the last decade. One of the first proposals for representing variability models of system families is by using modal transition systems (MTSs) [28], where optional 'may' transitions are used to model variability. In contrast, here we use MTSs with an entirely different goal of abstracting variability models, which is closer to the original idea of introducing MTSs by Larsen and Thomsen [20]. Subsequently, Classen et al. [4] present featured transition systems (FTSs), which are today widely accepted as the model essentially sufficient for most purposes of lifted model checking. They show how specifically designed lifted model checking algorithms (implemented in ProVeLines [29]) can be used for verifying FTSs against LTL properties. Classen et al. [18] also present symbolic lifted model checking algorithms (implemented as an extension of NuSMV model checker) for verifying FTSs against CTL properties.

The variability abstractions and the corresponding abstract variability models that preserve LTL are introduced in [6, 7]. Subsequently, automatic abstraction-refinement procedures for lifted model checking of LTL are proposed [30, 31], which use Craig interpolation to define the refinement. If a spurious counterexample (introduced due to the abstraction) is found in the abstract model, the procedures [30, 31] use Craig interpolation to extract relevant information from it in order to define the refinement of abstract models. The first author has previously introduced variability abstractions that preserve all (universal and existential) CTL properties [9], but without an automatic mechanism for constructing them and no notion of refinement. The abstractions have to be constructed manually before verification. In order to make the entire verification procedure automatic, we develop here an abstraction and refinement framework for CTL properties by using model checking games to define the refinement [15]. In this paper, we explain this abstraction-refinement procedure for CTL lifted model checking in details and we further pursue this line of work by employing the notion of hyper-transitions in order to give a generalized definition of abstract models. In this way, we obtain a more effective verification procedure. Moreover, since abstract variability models also preserve the full μ-calculus [9] and the game-based model checking for μ-calculus is defined [27, 14], we adapt our abstraction-refinement procedure for verifying μ-calculus properties.

Another approach to efficiently verify variability models is by using variability encoding [32], which transforms features into non-deterministically

initialized variables (replaces compile-time with run-time variability). The generated *family simulator* is verified using the standard single-system model checkers. However, in case of violation, the (single-system) model checker stops after a single counterexample and a violating variant are found. Therefore, this answer is incomplete (limited) since there might be other satisfying variants and also there might be other violating variants with different counterexamples. In contrast, lifted model checking and our approach provide precise conclusive results for all variants.

One of the earliest attempts for using games for CTL model checking has been proposed by Stirling [12]. Shoham and Grumberg [13, 14] have extended this game-based approach for defining compositional abstraction-refinement framework for CTL over 3-valued semantics. Subsequently, Shoham and Grumberg [16] introduced the notion of hyper-transitions in order to give a more generalized definition of abstract models of TSs that preserve CTL. This gives rise to a monotonic abstraction-refinement framework for full CTL. In this paper, we employ Shoham&Grumberg's algorithm for game-based model checking and the notion of hyper-transition in an entirely new context of lifted model checking. Thus, we establish a brand new connection between games and lifted (SPL) model checking.

Variability abstractions have also been employed in lifted static analysis [33]. They aim to tame the combinatorial explosion of the number of configurations and reduce it to something more tractable by manipulating the configuration space. Such variability abstractions are used for deriving abstract lifted static analyses, which enable deliberate trading of precision for speed. A technique for automatic generation of suitable variability abstractions for lifted static analysis is presented in [34]. It uses a pre-analysis to estimate the impact of variability-specific parts of the program family on analysis's precision. The obtained results from running the pre-analysis are used for constructing a suitable abstract lifted static analysis.

10. Conclusion

In this work we present a game-based lifted model checking for abstract variability models with respect to the full CTL. We also suggest an automatic refinement procedure, in case the model checking result is indefinite. We use the indefinite part of the colored game-graph of an abstract model to derive a failure node and a reason, which are then exploited for refinement. Moreover, we generalize the definition of abstract variability models of FTSs

by using the notion of hyper-transitions. This results in more precise abstract models in which more CTL properties can be proved or disproved. Finally, we suggest an automatic generalized abstraction-refinement procedure, in case the model checking result is indefinite. We also show how to adapt our abstraction-refinement procedures for verifying μ-calculus properties.

References

[1] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley, 2001.

[2] K. Pohl, G. Böckle, F. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques, Springer, 2005.

[3] C. Ebert, C. Jones, Embedded software: Facts, figures, and future, IEEE Computer 42 (4) (2009) 42–52. doi:10.1109/MC.2009.118.
URL https://doi.org/10.1109/MC.2009.118

[4] A. Classen, M. Cordy, P. Schobbens, P. Heymans, A. Legay, J. Raskin, Featured transition systems: Foundations for verifying variability-intensive systems and their application to LTL model checking, IEEE Trans. Software Eng. 39 (8) (2013) 1069–1089. doi:10.1109/TSE.2012.86.
URL http://doi.ieeecomputersociety.org/10.1109/TSE.2012.86

[5] A. Classen, M. Cordy, P. Heymans, A. Legay, P. Schobbens, Model checking software product lines with SNIP, STTT 14 (5) (2012) 589–612. doi:10.1007/s10009-012-0234-1.
URL http://dx.doi.org/10.1007/s10009-012-0234-1

[6] A. S. Dimovski, A. S. Al-Sibahi, C. Brabrand, A. Wasowski, Family-based model checking without a family-based model checker, in: Model Checking Software - 22nd International Symposium, SPIN 2015, Proceedings, Vol. 9232 of LNCS, Springer, 2015, pp. 282–299. doi:10.1007/978-3-319-23404-5_18.
URL http://dx.doi.org/10.1007/978-3-319-23404-5_18

[7] A. S. Dimovski, A. S. Al-Sibahi, C. Brabrand, A. Wasowski, Efficient family-based model checking via variability abstractions, STTT 19 (5) (2017) 585–603. doi:10.1007/s10009-016-0425-2.
URL https://doi.org/10.1007/s10009-016-0425-2

[8] A. S. Dimovski, A. Wasowski, From transition systems to variability models and from lifted model checking back to UPPAAL, in: Models, Algorithms, Logics and Tools - Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday, Vol. 10460 of LNCS, Springer, 2017, pp. 249–268. doi:10.1007/978-3-319-63121-9_13.
URL https://doi.org/10.1007/978-3-319-63121-9_13

[9] A. S. Dimovski, Abstract family-based model checking using modal featured transition systems: Preservation of ctl*, in: Fundamental Approaches to Software Engineering - 21st International Conference, FASE 2018, Proceedings, Vol. 10802 of LNCS, Springer, 2018, pp. 301–318.

[10] A. S. Dimovski, {CTL*} family-based model checking using variability abstractions and modal transition systems, STTT 22 (1) (2020) 35–55. doi:10.1007/s10009-019-00528-0.
URL https://doi.org/10.1007/s10009-019-00528-0

[11] E. M. Clarke, E. A. Emerson, Design and synthesis of synchronization skeletons using branching-time temporal logic, in: Logics of Programs, Workshop, 1981, Vol. 131 of LNCS, Springer, 1981, pp. 52–71. doi:10.1007/BFb0025774.
URL https://doi.org/10.1007/BFb0025774

[12] C. Stirling, Modal and Temporal Properties of Processes, Texts in Computer Science, Springer, 2001. doi:10.1007/978-1-4757-3550-5.
URL https://doi.org/10.1007/978-1-4757-3550-5

[13] S. Shoham, O. Grumberg, A game-based framework for CTL counterexamples and 3-valued abstraction-refinement, ACM Trans. Comput. Log. 9 (1) (2007) 1. doi:10.1145/1297658.1297659.
URL http://doi.acm.org/10.1145/1297658.1297659

[14] S. Shoham, O. Grumberg, Compositional verification and 3-valued abstractions join forces, Inf. Comput. 208 (2) (2010) 178–202. doi:10.1016/j.ic.2009.10.002.
URL https://doi.org/10.1016/j.ic.2009.10.002

[15] A. S. Dimovski, A. Legay, A. Wasowski, Variability abstraction and refinement for game-based lifted model checking of full CTL, in: Fundamental Approaches to Software Engineering - 22nd International Conference,

FASE 2019, Proceedings, Vol. 11424 of LNCS, Springer, 2019, pp. 192–209. doi:10.1007/978-3-030-16722-6_11.
URL https://doi.org/10.1007/978-3-030-16722-6_11

[16] S. Shoham, O. Grumberg, Monotonic abstraction-refinement for CTL, in: Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Proceedings, Vol. 2988 of LNCS, Springer, 2004, pp. 546–560. doi:10.1007/978-3-540-24730-2_40.
URL https://doi.org/10.1007/978-3-540-24730-2_40

[17] K. G. Larsen, X. Liu, Equation solving using modal transition systems, in: Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), IEEE Computer Society, 1990, pp. 108–117. doi:10.1109/LICS.1990.113738.
URL https://doi.org/10.1109/LICS.1990.113738

[18] A. Classen, P. Heymans, P.-Y. Schobbens, A. Legay, Symbolic model checking of software product lines, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, ACM, 2011, pp. 321–330. doi:10.1145/1985793.1985838.
URL http://doi.acm.org/10.1145/1985793.1985838

[19] C. Baier, J. Katoen, Principles of model checking, MIT Press, 2008.

[20] K. G. Larsen, B. Thomsen, A modal process logic, in: Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS '88), IEEE Computer Society, 1988, pp. 203–210. doi:10.1109/LICS.1988.5119.
URL http://dx.doi.org/10.1109/LICS.1988.5119

[21] P. Godefroid, R. Jagadeesan, On the expressiveness of 3-valued models, in: Verification, Model Checking, and Abstract Interpretation, 4th International Conference, VMCAI 2003, Proceedings, Vol. 2575 of LNCS, Springer, 2003, pp. 206–222. doi:10.1007/3-540-36384-X_18.
URL https://doi.org/10.1007/3-540-36384-X_18

[22] P. Cousot, Partial completeness of abstract fixpoint checking, in: Abstraction, Reformulation, and Approximation, 4th International Symposium, SARA 2000, Proceedings, Vol. 1864 of LNCS, Springer, 2000, pp. 1–25. doi:10.1007/3-540-44914-0_1.
URL https://doi.org/10.1007/3-540-44914-0_1

[23] M. Plath, M. Ryan, Feature integration using a feature construct, Sci. Comput. Program. 41 (1) (2001) 53–84. doi:10.1016/S0167-6423(00)00018-6.
URL https://doi.org/10.1016/S0167-6423(00)00018-6

[24] S. Ben-David, B. Sterin, J. M. Atlee, S. Beidu, Symbolic model checking of product-line requirements using sat-based methods, in: 37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Volume 1, IEEE Computer Society, 2015, pp. 189–199. doi:10.1109/ICSE.2015.40.
URL https://doi.org/10.1109/ICSE.2015.40

[25] A. S. Dimovski, A. Legay, A. Wasowski, Variability abstraction and refinement for game-based lifted model checking of full ctl (extended version), CoRR abs/1902.05594 (2019).
URL http://arxiv.org/abs/1902.05594

[26] D. Kozen, Results on the propositional mu-calculus, Theor. Comput. Sci. 27 (1983) 333–354. doi:10.1016/0304-3975(82)90125-6.
URL https://doi.org/10.1016/0304-3975(82)90125-6

[27] O. Grumberg, M. Lange, M. Leucker, S. Shoham, *Don't Know* in the μ-calculus, in: Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Proceedings, Vol. 3385 of LNCS, Springer, 2005, pp. 233–249. doi:10.1007/978-3-540-30579-8_16.
URL https://doi.org/10.1007/978-3-540-30579-8_16

[28] K. G. Larsen, U. Nyman, A. Wasowski, Modal I/O automata for interface and product line theories, in: Programming Languages and Systems, 16th European Symposium on Programming, ESOP 2007, Proceedings, Vol. 4421 of LNCS, Springer, 2007, pp. 64–79. doi:10.1007/978-3-540-71316-6_6.
URL http://dx.doi.org/10.1007/978-3-540-71316-6_6

[29] M. Cordy, A. Classen, P. Heymans, P. Schobbens, A. Legay, Provelines: a product line of verifiers for software product lines, in: 17th International SPLC 2013 workshops, ACM, 2013, pp. 141–146. doi:10.1145/2499777.2499781.
URL http://doi.acm.org/10.1145/2499777.2499781

[30] M. Cordy, P. Heymans, A. Legay, P. Schobbens, B. Dawagne, M. Leucker, Counterexample guided abstraction refinement of product-line behavioural models, in: S. Cheung, A. Orso, M. D. Storey (Eds.), Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), ACM, 2014, pp. 190–201. doi:10.1145/2635868.2635919.
URL http://doi.acm.org/10.1145/2635868.2635919

[31] A. S. Dimovski, A. Wasowski, Variability-specific abstraction refinement for family-based model checking, in: Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Proceedings, Vol. 10202 of LNCS, 2017, pp. 406–423. doi:10.1007/978-3-662-54494-5_24.
URL http://dx.doi.org/10.1007/978-3-662-54494-5_24

[32] S. Apel, A. von Rhein, P. Wendler, A. Größlinger, D. Beyer, Strategies for product-line verification: case studies and experiments, in: 35th International Conference on Software Engineering, ICSE '13, IEEE Computer Society, 2013, pp. 482–491.

[33] A. S. Dimovski, C. Brabrand, A. Wasowski, Variability abstractions: Trading precision for speed in family-based analyses, in: 29th European Conference on Object-Oriented Programming, ECOOP 2015, Vol. 37 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 247–270. doi:10.4230/LIPIcs.ECOOP.2015.247.
URL http://dx.doi.org/10.4230/LIPIcs.ECOOP.2015.247

[34] A. S. Dimovski, C. Brabrand, A. Wasowski, Finding suitable variability abstractions for lifted analysis, Formal Asp. Comput. 31 (2) (2019) 231–259. doi:10.1007/s00165-019-00479-y.
URL https://doi.org/10.1007/s00165-019-00479-y