A Game-Based Framework for CTL Counterexamples and 3-Valued Abstraction-Refinement⋆

Sharon Shoham and Orna Grumberg

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

Abstract. This work exploits and extends the game-based framework of CTL model checking for counterexample and incremental abstraction-refinement. We define a game-based CTL model checking for abstract models over the 3-valued semantics, which can be used for verification as well as refutation. The model checking may end with an indefinite result, in which case we suggest a new notion of refinement, which eliminates indefinite results of the model checking. This provides an iterative abstraction-refinement framework. It is enhanced by an *incremental* algorithm, where refinement is applied only where indefinite results exist and definite results from prior iterations are used within the model checking algorithm. We also define the notion of *annotated counterexamples*, which are sufficient and minimal counterexamples for full CTL. We present an algorithm that uses the game board of the model checking game to derive an *annotated counterexample* in case the examined system model refutes the checked formula.

1 Introduction

This work exploits and extends the game-based framework [31] of CTL model checking for counterexample and incremental abstraction-refinement.

The first goal of this work is to suggest a game-based new model checking algorithm for the branching-time temporal logic CTL [7] in the context of abstraction. Model checking is a successful approach for verifying whether a system model M satisfies a specification φ, written as a temporal logic formula. Yet, concrete (regular) models of realistic systems tend to be very large, resulting in the *state explosion problem*. This raises the need for abstraction. Abstraction hides some of the system details, thus resulting in smaller models.

Two types of semantics are available for interpreting CTL formulae over abstract models. The *2-valued* semantics defines a formula φ to be either true or false in an abstract model. True is guaranteed to hold for the concrete model as well, whereas false may be spurious. The *3-valued* semantics [14] introduces a new truth value: the value of a formula on an abstract model may be *indefinite*, which gives no information on its value on the concrete model. On the other hand, both satisfaction and falsification w.r.t the 3-valued semantics hold for the concrete model as well. That is, abstractions over 3-valued semantics are conservative w.r.t. both positive and negative results. They thus give precise results more often both for verification and falsification.

⋆ A fuller version appears in http://www.cs.technion.ac.il/users/orna/publications.html

Following the above observation, we define a game-based model checking algorithm for abstract models w.r.t. the 3-valued semantics, where the abstract model can be used for both verification and falsification. However, a third case is now possible: model checking may end with an indefinite answer. This is an indication that our abstraction cannot determine the value of the checked property in the concrete model and therefore needs to be refined. The traditional abstraction-refinement framework [19,6] is designed for 2-valued abstractions, where false may be a false-alarm, thus refinement is aimed at eliminating false results. As such, it is usually based on a counterexample analysis. Unlike this approach, the goal of our refinement is to eliminate indefinite results and turn them into either definite true or definite false.

An advantage of this work lies in the fact that the refinement is then applied only to the indefinite part of the model. Thus, the refined abstract model does not grow unnecessarily. In addition, model checking of the refined model uses definite results from previous runs, resulting in an *incremental* model checking. Our abstraction-refinement process is complete in the sense that for a finite concrete model it will always terminate with a definite "yes" or "no" answer.

The next goal of our work is to use the game-based framework in order to provide counterexamples for full CTL. When model checking a model M with respect to a property φ, if M does not satisfy φ then the model checker tries to return a counterexample. Typically, a counterexample is a part of the model that demonstrates the reason for the refutation of φ on M. Providing counterexamples is an important feature of model checking which helps tremendously in the debugging of the verified system.

Most existing model checking tools return as a counterexample either a finite path (for refuting formulae of the form AGp) or a finite path followed by a cycle (for refuting formulae of the form AFp[1]) [5,7]. Recently, this approach has been extended to provide counterexamples for all formulae of the universal branching-time temporal logic ACTL [9]. In this case the part of the model given as the counterexample has the form of a tree. Other works also extract information from model checking [29,12,25,32]. Yet, it is presented in the form of a temporal proof, rather than a part of the model.

In this work we provide counterexamples for full CTL. As for ACTL, counterexamples are part of the model. However, when CTL is considered, we face existential properties as well. To prove refutation of an existential formula $E\psi$, one needs to show an initial state from which *all* paths do not satisfy ψ. Thus, the structure of the counterexample becomes more complex. Having such a complex counterexample, it might not be easy for the user to analyze it by looking at the subgraph of M alone. We therefore *annotate* each state on the counterexample with a subformula of φ that is false in that state. The annotating subformulae being false in the respective states, provide the reason for φ to be false in the initial state. Thus, the annotated counterexample gives a convenient tool for debugging. We propose an algorithm that constructs an annotated counterexample and prove that it is sufficient and minimal.

To conclude, the main contributions of this work are:

– A game-based CTL model checking for abstract models over the 3-valued semantics, which can be used for verification as well as refutation.

---

[1] AGp means "for every path, in every state on the path, p holds", whereas AFp means "along every path there is a state which satisfies p".

- – A new notion of refinement, that eliminates indefinite results of the model checking.
- – An incremental model checking within the framework of abstraction-refinement.
- – A sufficient and minimal counterexample for full CTL.

Related Work. Other researchers have suggested abstraction-refinement mechanisms for various branching time temporal logics. In [21] the tearing paradigm is presented as a way to obtain lower and upper approximations of the system . Yet, their technique is restricted to ACTL or ECTL. In [27,28] the full propositional mu-calculus is considered. In their abstraction, the concrete and abstract systems share the same state space. The simplification is based on taking supersets and subsets of a given set with a more compact BDD representation. In [23] full CTL is handled. However, the verified system has to be described as a cartesian product of machines. The initial abstraction considers only machines that directly influence the formula and in each iteration the cone of influence is extended in a BFS manner. [1] handles ACTL and full CTL. Their abstraction collapses all states that satisfy the same subformulae of φ into an abstract state. Thus, computing the abstract model is at least as hard as model checking. Instead, they use partial knowledge on the abstraction function and gain information in each refinement.

Other researchers [14] have suggested to evaluate a property w.r.t the 3-valued semantics by reducing the problem to two 2-valued model checking problems: one for satisfaction and one for refutation. Such a reduction results in the same answer as our algorithm. Yet, it is then not clear how to guide the refinement, in case it is needed.

The game-based approach to model checking, used in this work, is closely related to the Automata-theoretic approach [18], as described in [22]. Thus, our work can also be described in this framework, using alternating automata.

Organization. The rest of the paper is organized as follows. In Section 2 we give some background for game-based CTL model checking, abstractions and the 3-valued semantics. Due to technical reasons, we then start with annotated counterexamples. In Section 3 we describe how to construct an annotated counterexample for full CTL and show that it is sufficient and minimal. In Section 4 we extend the game-based model checking to abstract models using the 3-valued semantics. In Section 5 we present our refinement technique, as well as an incremental abstraction-refinement framework.

2   Preliminaries

Let AP be a finite set of atomic propositions. We define the set *Lit* of literals over AP to be the set $AP \cup \{\neg p : p \in AP\}$. We identify $\neg\neg p$ with p. In this paper we consider the logic CTL in *negation normal form*, defined as follows: $\varphi ::= \text{tt} \mid \text{ff} \mid l \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid A\psi \mid E\psi$ where l ranges over Lit, and ψ is defined by $\psi ::= X\varphi \mid \varphi U\varphi \mid \varphi V\varphi$.

The (concrete) semantics of CTL formulae is defined with respect to a *Kripke structure (KS)* $M = (S, S_0, \rightarrow, L)$, where S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $\rightarrow \subseteq S \times S$ is a transition relation, which must be *total* and $L : S \rightarrow 2^{Lit}$ is a labeling function, such that for every state s and every $p \in AP$, $p \in L(s)$ iff $\neg p \notin L(s)$.

$[(M, s) \models \varphi] = \text{tt} (= \text{ff})$ means that the CTL formula φ is true (false) in state s of a KS M. The formal definition can be found in [7]. M *satisfies* φ, denoted $[M \models \varphi] = \text{tt}$, if $\forall s_0 \in S_0 : [(M, s_0) \models \varphi] = \text{tt}$. Otherwise, M *refutes* φ, denoted $[M \models \varphi] = \text{ff}$.

Definition 1. *Given a CTL formula* φ *of the form* $Q(\varphi_1 U \varphi_2)$ *or* $Q(\varphi_1 V \varphi_2)$*, where* $Q \in \{A, E\}$*, its* expansion *is defined by:*
if $\varphi = Q(\varphi_1 U \varphi_2)$ *then* $exp(\varphi) = \{\varphi,\ \varphi_2 \vee (\varphi_1 \wedge QX\varphi),\ \varphi_1 \wedge QX\varphi,\ QX\varphi\}$
if $\varphi = Q(\varphi_1 V \varphi_2)$ *then* $exp(\varphi) = \{\varphi,\ \varphi_2 \wedge (\varphi_1 \vee QX\varphi),\ \varphi_1 \vee QX\varphi,\ QX\varphi\}$

2.1 Game-Based Model Checking Algorithm

Given a (concrete) KS $M = (S, S_0, \rightarrow, L)$ and a CTL formula φ, the *model checking game* [31,22] of M and φ is defined as follows. Its board is $S \times sub(\varphi)$, where $sub(\varphi)$ is the set of subformulae of φ, defined as usual, except that if $\varphi = A(\varphi_1 U \varphi_2)$, $E(\varphi_1 U \varphi_2)$, $A(\varphi_1 V \varphi_2)$ or $E(\varphi_1 V \varphi_2)$ then $sub(\varphi) = exp(\varphi) \cup sub(\varphi_1) \cup sub(\varphi_2)$.

The model checking game is played by two players, $\forall$belard, the refuter, and $\exists$loise, the prover. A *play* is a (possibly infinite) sequence $C_0 \rightarrow_{p0} C_1 \rightarrow_{p1} C_2 \rightarrow_{p2} \ldots$ of configurations, where $C_0 \in S_0 \times \{\varphi\}$, $C_i \in S \times sub(\varphi)$ and $p_i \in \{\forall, \exists\}$. The subformula in C_i determines which player p_i makes the next move.

Possible Moves at Each Step

1. $C_i = (s, \text{ff})$, $C_i = (s, \text{tt})$, or $C_i = (s, l)$ where $l \in Lit$: the play is finished. Such configurations are called *terminal configurations*.
2. $C_i = (s, AX\varphi)$: $\forall$belard chooses a transition $s \rightarrow s'$ and $C_{i+1} = (s', \varphi)$.
3. $C_i = (s, EX\varphi)$: $\exists$loise chooses a transition $s \rightarrow s'$ and $C_{i+1} = (s', \varphi)$.
4. $C_i = (s, \varphi_1 \wedge \varphi_2)$: $\forall$belard chooses $j \in \{1, 2\}$ and $C_{i+1} = (s, \varphi_j)$.
5. $C_i = (s, \varphi_1 \vee \varphi_2)$: $\exists$loise chooses $j \in \{1, 2\}$ and $C_{i+1} = (s, \varphi_j)$.
6. $C_i = (s, Q(\varphi_1 U \varphi_2))$, $Q \in \{A, E\}$: $C_{i+1} = (s, \varphi_2 \vee (\varphi_1 \wedge QXQ(\varphi_1 U \varphi_2)))$.
7. $C_i = (s, Q(\varphi_1 V \varphi_2))$, $Q \in \{A, E\}$: $C_{i+1} = (s, \varphi_2 \wedge (\varphi_1 \vee QXQ(\varphi_1 V \varphi_2)))$.

In configurations 6-7 the move is deterministic, thus any player can make the move. A play is *maximal* iff it is infinite or ends in a terminal configuration. In [31] it is shown that a play is infinite iff exactly one subformula of the form AU, EU, AV or EV occurs in it infinitely often. Such a subformula is called a *witness*.

Winning Criteria. $\forall$belard wins the play iff (1) the play is finite and ends in a terminal configuration of the form $C_i = (s, \text{ff})$, or $C_i = (s, l)$, where $l \notin L(s)$, or (2) the play is infinite and the witness is AU or EU. $\exists$loise wins otherwise.

The model checking *game* consists of all the possible plays. A *winning strategy* is a set of rules for a player, telling him how to move in the current configuration and allowing him to win every play if he plays by the rules. All possible plays of a game are captured in the *game-graph*. It is the graph whose nodes are the elements (configurations) of the game board and whose edges are the possible moves of the players.

The model checking algorithm for the evaluation of $[M \models \varphi]$ consists of two parts. First, it constructs (part of) the *game-graph*. The evaluation of the truth value of φ in M is then done by coloring the game-graph.

Game-Graph Construction and Its Properties. The subgraph of the game-graph that is reachable from the initial configurations $S_0 \times \{\varphi\}$ is constructed in a BFS or DFS manner. It is denoted $G_{M \times \varphi} = (N, E)$, where $N \subseteq S \times sub(\varphi)$. The nodes (configurations) of $G_{M \times \varphi}$ can be classified into three types. Terminal configurations are *leaves*

in the game-graph. Nodes whose subformulae are of the form $\varphi_1 \wedge \varphi_2$ or $AX\varphi_1$ are $\wedge$-*nodes*. Nodes whose subformulae are of the form $\varphi_1 \vee \varphi_2$ or $EX\varphi_1$ are $\vee$-*nodes*. Nodes whose subformulae are AU, EU, AV, EV can be considered either $\vee$-nodes or $\wedge$-nodes. Sometimes we further distinguish between nodes whose subformulae are of the form $AX\varphi$ ($EX\varphi$) and other $\wedge$-nodes ($\vee$-nodes), by referring to them as *AX-nodes* (*EX-nodes*). The edges in $G_{M \times \varphi}$ are divided to *progress edges*, that originate in AX-nodes or EX-nodes and reflect transitions of the KS, and *auxiliary edges*, which are the rest. Each non-trivial *strongly connected component* (SCC) in $G_{M \times \varphi}$, i.e. an SCC with one edge at least, contains exactly one witness and is classified as an AU, AV, EU, or EV SCC, based on its witness.

Coloring Algorithm. The following *Coloring Algorithm* [3] labels each node in $G_{M \times \varphi}$ by T or F, depending on whether $\exists$loise or $\forall$belard has a winning strategy. $G_{M \times \varphi}$ is partitioned into its *Maximal Strongly Connected Components* (MSCCs), denoted Q_i's, and an order $\leq$ is determined on them, such that an edge (n, n'), where $n \in Q_i$ and $n' \in Q_j$, exists in $G_{M \times \varphi}$ only if $Q_j \leq Q_i$. Such an order exists because the MSCCs form a *directed acyclic graph* (DAG). It can be 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 MSCC w.r.t $\leq$ that is not yet fully colored. Every outgoing edge of Q_i leads to a colored node or remains in the same set, Q_i. The nodes of Q_i are colored as follows.

1. Terminal nodes are colored by T if $\exists$loise wins in them, and by F otherwise.
2. An $\vee$-node ($\wedge$-node) is colored by T (F) if it has a son that is colored by T (F), and by F (T) if all its sons are colored by F (T).
3. All the nodes in Q_i that remain uncolored, after the propagation of these rules, are colored according to the witness in Q_i. They are colored by F if the witness is of the form AU or EU, and are colored by T if the witness is of the form AV or EV.

The result of the coloring algorithm is a *coloring function* $\chi : N \to \{T, F\}$.

Theorem 1. *[31] Let M be a KS, φ a CTL formula and $(s, \varphi_1) \in G_{M \times \varphi}$. Then:*
$[(M, s) \models \varphi_1] = tt$ *(ff)* $\Leftrightarrow$ $\exists$loise ($\forall$belard) has a winning strategy for the game starting at (s, φ_1) $\Leftrightarrow$ (s, φ_1) is colored by T (F).

2.2 Abstraction

Abstract models preserving CTL need to have two transition relations [20,11]. This is achieved by using *Kripke Modal Transition Systems* [17,13].

Definition 2. *A* Kripke Modal Transition System *(KMTS) is a tuple* $M = (S, S_0, \overset{must}{\longrightarrow}, \overset{may}{\longrightarrow}, L)$*, where S is a finite set of states, $S_0 \subseteq S$ is a set of initial states,* $\overset{must}{\longrightarrow} \subseteq S \times S$ *and* $\overset{may}{\longrightarrow} \subseteq S \times S$ *are transition relations such that* $\overset{must}{\longrightarrow} \subseteq \overset{may}{\longrightarrow}$*, and $L : S \to 2^{Lit}$ is a labeling function, s.t. for each state s and $p \in AP$, at most one of p and $\neg p$ is in $L(s)$.*

We consider abstractions that collapse sets of concrete states into single abstract states. Such abstractions can be described in the framework of *Abstract Interpretation* [24,11]. Let $M_C = (S_C, S_{0C}, \to, L_C)$ be a (concrete) KS. Let $(S_A, \sqsubseteq)$ be a poset of *abstract states* and $(\gamma : S_A \to 2^{S_C}, \alpha : 2^{S_C} \to S_A)$ a *Galois connection* [10,24]

from $(2^{S_C}, \subseteq)$ to $(S_A, \sqsubseteq)$. γ is the *concretization function* that maps each abstract state to the set of concrete states that it represents. α is the *abstraction function* that maps each set of concrete states to the abstract state that represents it.

An abstract model M_A can then be defined as follows. The set of initial abstract states S_{0A} is defined such that $s_{0a} \in S_{0A}$ iff there exists $s_{0c} \in S_{0C}$ for which $s_{0c} \in \gamma(s_{0a})$. An abstract state s_a is labeled by $l \in Lit$ only if all the concrete states that it represents are labeled by l. Thus, it is possible that neither p nor $\neg p$ are in $L_A(s_a)$. The *may*-transitions are computed s.t. they represent (at least) every concrete transition: if $\exists s_c \in \gamma(s_a)$ and $\exists s'_c \in \gamma(s'_a)$ s.t. $s_c \to s'_c$, then $s_a \xrightarrow{\text{may}} s'_a$. The *must*-transitions represent concrete transitions that are common to all the concrete states represented by the origin abstract state: $s_a \xrightarrow{\text{must}} s'_a$ only if $\forall s_c \in \gamma(s_a) \exists s'_c \in \gamma(s'_a)$ s.t. $s_c \to s'_c$. Other constructions of abstract models, based on Galois connections, can be found in [11,15].

The relation $H \in S_C \times S_A$, which is defined by $(s_c, s_a) \in H$ iff $s_c \in \gamma(s_a)$, then forms a *mixed simulation* [11,13] from M_C to the resulting abstract model M_A.

[17] defines the *3-valued* semantics of CTL over KMTSs, denoted $[(M, s) \overset{3}{\models} \varphi]$, preserving both satisfaction (tt) and refutation (ff) from the abstract model to the concrete one. However, a new truth value, $\bot$, is introduced, meaning that the truth value over the concrete model is not known and can be either tt or ff.

Theorem 2. *[13] Let $H \subseteq S_C \times S_A$ be a mixed simulation relation from a KS M_C to a KMTS M_A. Then for every $(s_c, s_a) \in H$ and every CTL formula φ:*
$$[(M_A, s_a) \overset{3}{\models} \varphi] = tt \ (\text{ff}) \ \Rightarrow \ [(M_C, s_c) \models \varphi] = tt \ (\text{ff})$$

3   Using Games to Produce Annotated Counterexamples

In this section we describe how to construct an *annotated counterexample* from the coloring of a game-graph for M and φ in case M does not satisfy φ.

First, the coloring algorithm is changed to identify and remember the *cause* of the coloring of an $\wedge$-node n that is colored by F. If n was colored by its sons, then $cause(n)$ is the son that was first to be colored by F. If n was colored due to a witness, then $cause(n)$ is chosen to be one of its sons which resides on the same SCC and was colored by witness as well. There must exist such a son, otherwise n would be colored by its sons. Note that $cause(n)$ depends on the execution of the coloring algorithm.

Given a game-graph $G_{M \times \varphi}$, for a KS M and a CTL formula φ, and given its coloring χ and an initial node $n_0 = (s_0, \varphi)$ s.t. $\chi(n_0) = F$, the following algorithm finds an *annotated counterexample*, denoted C, which is a subgraph of $G_{M \times \varphi}$ colored by F.

Algorithm `ComputeCounter`
Initially: $new = \{(s_0, \varphi)\}$, $C = \emptyset$.
while $new \neq \emptyset$
   n = remove (new)
   - if n was already handled or if n is a terminal node - continue.
   - if n is an $\vee$-node, then for each son n' of n add n' to new and (n, n') to C.
   - if n is an $\wedge$-node, then add $cause(n)$ to new and $(n, cause(n))$ to C.

Complexity. Algorithm `ComputeCounter` has a linear running time (in the worst case) w.r.t the size of the game-graph $G_{M \times \varphi}$. The latter is bounded by $O(|M| \cdot |\varphi|)$.

Note, that for the correctness of C it is *mandatory* to choose for an $\wedge$-node the son that caused the coloring of the node, and not any son that was colored by F.

Properties of the Computed Annotated Counterexample. C is a subgraph of $G_{M\times\varphi}$ s.t. for each node $n \in C$, $\chi(n) = F$. It can be viewed as the part of the winning strategy of the refuter that is sufficient to guarantee his victory. We formalize and prove this notion in the next section. Intuitively, it is indeed a counterexample in the sense that it points out the reasons for φ's refutation on the model. Each node in C is marked by a state s and a subformula φ_1, s.t. $\chi((s,\varphi_1)) = F$, thus by Theorem 1, $[s \models \varphi_1] = \mathrm{ff}$. The edges point out the reason (cause) for the refutation of a certain subformula in a certain state: the refutation in an $\wedge$-node is shown by refutation in one of its sons, whereas the refutation in an $\vee$-node is shown by all its sons. Another important property is:

Lemma 1. *C contains non-trivial SCCs iff at least one of the nodes in the SCC was colored due to a witness. We conclude that non-trivial SCCs in C are AU- or EU-SCCs.*

The property of C described in Lemma 1 implies that any non-trivial SCC that appears in the annotated counterexample indicates a refutation of the U operator, which results, at least partly, from an infinite path, where weak until is satisfied, but not strong until.

3.1   The Annotated Counterexample Is Sufficient and Minimal

In this section we first informally describe our requirements of a counterexample. We then formalize them for annotated counterexamples and show that they are fullfilled by the result of `ComputeCounter`. Generally speaking, for a sub-model to be a counterexample, it is expected to: (1) falsify the given formula; (2) hold "enough" information to explain why the model refutes the formula; and (3) be minimal in the sense that removing any state or transition will not maintain 1 and 2. To formalize the second requirement w.r.t an annotated counterexample, we need the following definitions.

Definition 3. *Let $G = (N, E)$ be a game-graph and let A be a subgraph of G. The* partial coloring algorithm *of G w.r.t A works as follows. It is given an* initial coloring function $\chi_I : N \setminus A \rightarrow \{T, F\}$ *and computes a coloring function for G. The algorithm is identical to the (original) coloring algorithm, except for the addition of a new rule:*
   - A node $n \in N \setminus A$ is colored by $\chi_I(n)$ and its color is not changed.
Any result of the partial coloring algorithm of G with respect to A is called a partial coloring function *of G with respect to A, denoted $\overline{\chi} : N \rightarrow \{T, F\}$.*

As opposed to the usual coloring algorithm that has only one possible result, the partial coloring algorithm has several possible results, depending on the initial coloring function χ_I. Each one of them is considered a partial coloring function of G w.r.t A. By definition, the usual coloring algorithm is a partial coloring algorithm of G w.r.t G.

Definition 4. *Let G be a game-graph and let χ be the result of the coloring algorithm on G. A subgraph A of G is* independent of *G if for each $\overline{\chi}$ that is a partial coloring function of G with respect to A, and for each $n \in A$, we have that $\chi(n) = \overline{\chi}(n)$.*

Basically, a subgraph A is independent of G if its coloring is *absolute* in the sense that all of its completions to the full game-graph do not change the color of any node in A.

We can now formalize the notion of an annotated counterexample.

Definition 5. *Let G be a game-graph, and let χ be its coloring function, such that $\chi(n_0) = F$ for some initial node n_0. A subgraph $\tilde{C}$ of G containing n_0 is an* annotated counterexample *if it satisfies the following conditions. (1) For each node $n \in \tilde{C}$, $\chi(n) = F$; (2) $\tilde{C}$ is independent of G; and (3) $\tilde{C}$ is minimal.*

The first two requirements in Definition 5 imply that $\tilde{C}$ is *sufficient* for explaining why n_0 is colored F: First it guarantees that all the nodes in $\tilde{C}$ are colored F. In addition, since $\tilde{C}$ is independent of G, we can conclude that regardless of the other nodes in G, all the nodes in $\tilde{C}$, and in particular n_0, will be colored F. Thus, it also explains why the model falsifies the formula. The third condition shows that $\tilde{C}$ is also "necessary".

We now show that the result of `ComputeCounter`, denoted C, is indeed an annotated counterexample. The first requirement is obviously fulfilled, as described earlier. The following theorem states that C satisfies the other two conditions as well.

Theorem 3. *C is independent of G. Moreover, C is* minimal *in the sense that removing a node or an edge will result in a subgraph that is not independent of G.*

The correctness of the first part of Theorem 3 strongly depends on the choice of $cause(n)$ as the son of an $\wedge$-node in the algorithm `ComputeCounter`.

4 Game-Based Model Checking on Abstract Models

In this section we suggest a generalization of the game-based model checking algorithm for evaluating a CTL formula φ over a KMTS M w.r.t the 3-valued semantics.

We start with the description of the 3-valued game. The main difference arises from the fact that KMTSs have two types of transitions. Since the transitions of the model are considered only in configurations with subformulae of the form $AX\varphi_1$ or $EX\varphi_1$, these are the only cases where the rules of the play need to be changed. Intuitively, in order to be able to both prove and refute each subformula, the game needs to allow the players to use both may and must transitions in such configurations. The reason is that for example, truth of a formula $AX\varphi_1$ should be checked upon may-transitions, but its falseness should be checked upon must-transitions.

New Moves of the Game

2. if $C_i = (s, AX\varphi)$, then ∀belard chooses a transition $s \xrightarrow{must} s'$ (for refutation) or $s \xrightarrow{may} s'$ (for satisfaction), and $C_{i+1} = (s', \varphi)$.
3. if $C_i = (s, EX\varphi)$, then ∃loise chooses a transition $s \xrightarrow{must} s'$ (for satisfaction) or $s \xrightarrow{may} s'$ (for refutation), and $C_{i+1} = (s', \varphi)$.

Intuitively, the players use must-transitions in order to win, while they use may transitions in order to prevent the other player from winning. As a result it is possible that none of the players wins the play, i.e. the play ends with a tie. As before, a *maximal* play is infinite if and only if exactly one *witness*, which is either an AU, EU, AV or EV-formula, appears in it infinitely often. However, the winning rules become more complicated. A player can only win the play if he or she are "consistent" in their moves:

Definition 6. *A player is said to* play consistently *if in each configuration where he proceeds over the transitions of the model, his move is based on a $\xrightarrow{must}$ transition.*

New Winning Criteria

- $\forall$belard wins a play iff he plays consistently and in addition one of the following holds: (1) The play is finite and ends in a configuration $C_i = (s, \text{ff})$ or (s, l), where $\neg l \in L(s)$; or (2) The play is infinite and the witness is of the form AU or EU.
- $\exists$loise wins a play iff she plays consistently and in addition one of the following holds: (1) the play is finite and ends in configuration $C_i = (s, \text{tt})$ or (s, l), where $l \in L(s)$; or (2) the play is infinite and the witness is of the form AV or EV.
- Otherwise, the play ends with a tie.

Theorem 4. *Let M be a KMTS and φ a CTL formula. Then, for each $s \in S$:*

1. $[(M, s) \stackrel{3}{\models} \varphi] = tt$ *iff $\exists$loise has a winning strategy for the game starting at (s, φ).*
2. $[(M, s) \stackrel{3}{\models} \varphi] = ff$ *iff $\forall$belard has a winning strategy for the game starting at (s, φ).*
3. $[(M, s) \stackrel{3}{\models} \varphi] = \bot$ *iff none of them has a winning strategy for the game from (s, φ).*

In order to use this correspondence for model checking, we generalize the game-based model checking algorithm. The (3-valued) game-graph, denoted $G_{M \times \varphi}$, is constructed as in the "concrete" case. Its nodes, denoted N, are again classified as $\wedge$-nodes, $\vee$-nodes, AX-nodes or EX-nodes. Similarly, the edges are classified as *progress* edges or *auxiliary* edges. But now, we distinguish between two types of progress edges: Edges that are based on must-transitions are referred to as *must-edges*. Edges that are based on may-transitions are referred to as *may-edges*. A node n' is a *may-son* (*must-son*) of the node n if there exists a may-edge (must-edge) from n to n'. An SCC in the game-graph is a *may-SCC* (*must-SCC*) if all its progress edges are may-edges (must-edges).

The coloring algorithm of the 3-valued game-graph needs to be adapted as well. First, a new color, denoted ?, is introduced for configurations in which none of the players has a winning strategy. Second, the partition to Q_i's that is based on MSCCs is now based on may-MSCCs (note that $\xrightarrow{must} \subseteq \xrightarrow{may}$).

The (3-Valued) Coloring Algorithm

Partition and Order. $G_{M \times \varphi}$ is partitioned into its may-MSCCs, denoted $Q_i's$. A (total) order $\leq$ is determined on them in the same way as for the concrete case.

Coloring. As before, the coloring algorithm processes the Q_i's bottom-up. Let Q_i be the smallest set w.r.t $\leq$ that is not yet fully colored. Its nodes are colored in two phases.

1. *Sons-coloring phase.* Apply the following rules to Q_i until none is applicable.
   - A terminal node is colored by T if $\exists$loise wins in it, by F if $\forall$belard wins in it, and by ? otherwise.
   - An AX-node (EX-node) is colored by:
     - T (F) if all its may-sons are colored T (F).
     - F (T) if it has a must-son that is colored F (T).
     - ? if all its must sons are colored T (F) or ? and it has a may-son that is colored F (T) or ?.

- An $\wedge$-node ($\vee$-node), other than AX-node (EX-node), is colored by:
  - T (F) if both its sons are colored T (F).
  - F (T) if it has a son that is colored F (T).
  - ? if it has a son that is colored ? and the other is colored ? or T (F).

2. *Witness-coloring phase.* If after the propagation of the rules of phase 1 there are still uncolored nodes in Q_i, then Q_i must be a non-trivial may-MSCC that has exactly one witness. Its uncolored nodes are colored according to the witness, as follows.
   - The witness is of the form $A(\varphi_1 U \varphi_2)$ or $E(\varphi_1 U \varphi_2)$:
     - (a) Repeatedly color ? each node in Q_i satisfying one of the following.
       - An $\wedge$-node (AX-node) that all its (must) sons are colored T or ?.
       - An $\vee$-node (EX-node) that has a (may) son that is colored T or ?.
     - (b) Color the remaining nodes in Q_i by F.
   - The witness is of the form $A(\varphi_1 V \varphi_2)$ or $E(\varphi_1 V \varphi_2)$:
     - (a) Repeatedly color ? each node in Q_i satisfying one of the following.
       - An $\wedge$-node (AX-node) that has a (may) son that is colored F or ?.
       - An $\vee$-node (EX-node) that all its (must) sons are colored F or ?.
     - (b) Color the remaining nodes in Q_i by T.

The result of the coloring algorithm is a *3-valued coloring function* $\chi : N \to \{T, F, ?\}$. Note that a node is colored ? only if there is evidence that it cannot be colored otherwise.

Theorem 5. *Let $G_{M \times \varphi}$ be a 3-valued game-graph, then for each $n \in G_{M \times \varphi}$:*

1. $\chi(n) = T$ *iff* $\exists$*loise has a winning strategy for the game starting at n.*
2. $\chi(n) = F$ *iff* $\forall$*belard has a winning strategy for the game starting at n.*
3. $\chi(n) = ?$ *iff none of the players has a winning strategy for the game starting at n.*

The correctness of the coloring algorithm is strongly based on the property that when phase 2b is applied, the uncolored nodes that are colored in it form non-trivial SCCs. In case of an AU-witness, these are must-SCCs, and indeed in this case loops can only be used for refutation, thus to identify "real" loops, must-edges are needed. On the other hand, in case of an AV-witness, loops can contribute to satisfaction, and satisfaction of universal properties should be examined upon may-transitions, and indeed for such a witness, we get uncolored may-SCCs. Similarly, for an EU witness, we get may-SCCs, whereas for an EV witness, must-SCCs are formed.

Implementation Issues and Complexity. The coloring algorithm can be implemented in linear running time w.r.t the size of $G_{M \times \varphi}$, using a variation of an AND/OR graph, similarly to the algorithm described in [18] for checking nonemptiness of the language of a simple weak alternating word automaton. Thus, its running time is $O(|M| \cdot |\varphi|)$.

As a conclusion of Theorem 4 and Theorem 5, we get the following theorem.

Theorem 6. *Let M be a KMTS, φ a CTL formula and $(s, \varphi_1) \in G_{M \times \varphi}$. Then:*

$$[(M, s) \overset{3}{\models} \varphi_1] = \quad tt, ff \text{ or } \perp \quad \Leftrightarrow \quad (s, \varphi_1) \text{ is colored by } T, F \text{ or } ? \text{ respectively.}$$

Given the colored game-graph, if all the initial nodes are colored T, or if at least one of them is colored F, then by Theorem 6 and Theorem 2, there is a definite answer as for the satisfaction of φ in the *concrete* model. This is because there exists a mixed simulation from the concrete to the abstract model. Furthermore, if the result is ff, a *concrete* annotated counterexample can be produced, using an extension of `ComputeCounter`.

5   Refinement

In this section we show how to exploit the abstract game-graph in order to refine the abstract model in case model checking resulted in an indefinite answer. When the result is $\bot$, there is no reason to assume either one of the definite answers tt or ff. Thus, we would like to base the refinement not on a counterexample as in [19,6,2,8,4], but on the point(s) that are responsible for the uncertainty. The goal of the refinement is to discard these points, in the hope of getting a definite result on the refined abstraction.

Let $M_C = (S_C, S_{0C}, \rightarrow, L_C)$ be a concrete KS and let $M_A = (S_A, S_{0A}, \overset{must}{\longrightarrow}, \overset{may}{\longrightarrow}, L_A)$ be an abstract KMTS. Let $\gamma : S_A \rightarrow 2^{S_C}$ be the concretization function. Given the abstract 3-valued game-graph G, based on M_A, and its coloring function $\chi : N \rightarrow \{T, F, ?\}$, such that $\chi(n_0) = ?$ for some initial node n_0, we use the information gained by the coloring algorithm of G in order to refine the abstraction.

Refinement is done by splitting abstract states according to criteria obtained from *failure nodes*. A node is a *failure node* if it is colored by ?, whereas none of its sons was colored by ? at the time it got colored by the algorithm. Such a node is a failure node in the sense that it can be seen as the point where the loss of information occurred. Note, that a terminal node that is colored by ? is also considered a failure node. The coloring algorithm is adapted to remember failure nodes. In addition, for each node n that is colored by ?, but is *not* a failure node, the coloring algorithm remembers a son that was already colored ? by the time n was colored, denoted $cont(n)$.

Searching for a Failure Node. A *failure node* is found by a DFS-like greedy algorithm, starting from n_0: As long as the current node, n, is not a failure node, the algorithm proceeds to $cont(n)$. It ends and returns n when a failure node n is reached.

Lemma 2. *A failure node is either (1) a terminal node; (2) an AX-node (EX-node) that has a may-son colored by F (T); or (3) an AX-node (EX-node) that was colored during phase 2a based on an AU (EV) witness, and has a may-son colored by ?.*

Failure Analysis. Based on the failure node n, the refinement is reduced to the problem of separating sets of (concrete) states, which can be solved by known techniques, depending on the abstraction used (e.g. [8,6]). n provides the criterion for the separation:

1. $n = (s_a, l)$ is a terminal node. The reason for its indefinite color is that s_a represents both concrete states that are labeled by l and by $\neg l$. This is avoided by separating $\gamma(s_a)$ to two sets $\{s_c \in \gamma(s_a) : l \in L_C(s_c)\}$ and $\{s_c \in \gamma(s_a) : \neg l \in L_C(s_c)\}$.
2. $n = (s_a, AX\varphi_1)$ or $(s_a, EX\varphi_1)$ with a may-son colored F or T resp. Let K stand for F or T. We define $sons_K = \bigcup\{\gamma(s'_a) : (s'_a, \varphi_1)$ is a may son of n colored $K\}$ and $conc_K = \gamma(s_a) \cap \{s_c \in S_C : \exists s'_c \in sons_K, s_c \rightarrow s'_c\}$. For the $AX\varphi_1$ case, $K = F$ and $conc_K$ is the set of all concrete states, represented by s_a, that definitely refute $AX\varphi_1$. For the $EX\varphi_1$ case, $K = T$ and $conc_K$ is the set of all concrete states, represented by s_a, that definitely satisfy $EX\varphi_1$. In both cases, our goal is to separate the sets $conc_K$ and $\gamma(s_a) \setminus conc_K$.
3. $n = (s_a, AX\varphi_1)$ or $(s_a, EX\varphi_1)$ was colored during phase 2a based on an AU or an EV witness resp., and has a may-son $n' = (s'_a, \varphi_1)$ colored by ?. Let $conc_? = \gamma(s_a) \cap \{s_c \in S_C : \exists s'_c \in \gamma(s'_a), s_c \rightarrow s'_c\}$ be the set of all concrete states, represented by s_a, that have a son represented by s'_a. Our goal is to separate the sets $conc_?$ and $\gamma(s_a) \setminus conc_?$.

It is possible that one of the sets obtained during the failure analysis is empty and provides no criterion for the split. Yet, this is informative as well. As an example, consider case 2, where the failure node n is an AX-node. If $conc_F = \gamma(s_a)$, then every state represented by s_a has a refuting son. Thus, n can be colored F instead of ?. If $conc_F = \emptyset$, then none of the concrete states in $\gamma(s_a)$ has a transition to a concrete state represented by the F-colored may-sons of n. Thus, the may-edges from n to such sons can be removed. Either way, G can be recolored starting from the Q_i containing n.

The purpose of the split derived from cases 1-2 is to conclude definite results about (at least part) of the new abstract states obtained by the split of the failure node. These results can be used by the incremental algorithm, suggested below. As for case 3, we know that by the time the failure node n got colored, its may-son n' that is colored by ? was not yet colored (otherwise n would not be a failure node). By the description of the algorithm, if n' was a must-son of n, then as long as it was uncolored, n would remain uncolored too and would eventually be colored in phase 2b by a definite color. Thus, our goal in this case is to obtain a must edge between (parts of) n and n'.

Theorem 7. *For finite concrete models, iterating the abstraction-refinement process is guaranteed to terminate with a definite answer.*

5.1   Incremental Abstraction-Refinement Framework

We refine abstract models by splitting their states. The criterion for the refinement is decided locally, based on one node, but has a global effect. Yet, there is no reason to split states for which the model checking results are definite. The game-based model checking provides a convenient framework to use previous results, leading to an *incremental* model checking based on iterative abstraction-refinement, where each iteration consists of abstraction, model checking and refinement. After each iteration, we now remember the (abstract) nodes colored by definite colors, as well as nodes for which a definite color was discovered during failure analysis. When a refined game-graph is constructed, it is pruned in nodes that are *sub-nodes* of nodes remembered from previous iterations. A node (s_a, φ) is a *sub-node* of (s'_a, φ') if $\varphi = \varphi'$ and the concrete states represented by s_a are a subset of those represented by s'_a. Thus, only the reachable subgraph that was previously colored ? is refined. The coloring algorithm considers the nodes where the game-graph was pruned as leaves and colors them by their previous colors.

Note that for many abstractions, checking if a node is a sub-node of another is simple. For example, in the framework of predicate abstraction [16,30,26,15], this means that the abstract states "agree" on all the predicates that exist before the refinement.

References

1. A. Asteroth, C. Baier, and U. Assmann. Model checking with formula-dependent abstract models. In *Computer Aided Verification*, volume 2102 of *LNCS*, pages 155–168, 2001.
2. Sharon Barner, Daniel Geist, and Anna Gringauze. Symbolic localization reduction with reconstruction layering and backtracking. In *Computer Aided Verification*, 2002.

3. Benedikt Bollig, Martin Leucker, and Michael Weber. Local parallel model checking for the alternation-free mu-calculus. In *SPIN'02*. Springer-Verlag Inc., 2002.

4. P. Chauhan, E.M. Clarke, J. Kukula, S. Sapra, H. Veith, and D.Wang. Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis. In *Formal Methods in Computer Aided Design (FMCAD)*, November 2002.

5. E. Clarke, O. Grumberg, K. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *DAC'95*. IEEE Computer Society Press, 1995.

6. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, LNCS, Chicago, USA, July 2000.

7. E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT press, December 1999.

8. E.M. Clarke, A. Gupta, J. Kukula, and O. Strichman. SAT based abstraction-refinement using ILP and machine learning techniques. In *Computer-Aided Verification*, July 2002.

9. E.M. Clarke, S. Jha, Y. Lu, and H. Veith. Tree-like counterexamples in model checking. In *Seventeenth Annual IEEE Symposium on Logic In Computer Science (LICS)*, July 2002.

10. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *popl4*, pages 238–252, 1977.

11. Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(2), March 1997.

12. D.Peled, A.Pnueli, and L.Zuck. From falsification to verification. In *FSTTCS*, 2001.

13. P. Godefroid and R. Jagadeesan. Automatic abstraction using generalized model checking. In *Computer-Aided Verification*, volume 2404 of *LNCS*, pages 137–150, July 2002.

14. P. Godefroid and R. Jagadeesan. On the expressiveness of 3-valued models. In *Proc. of VMCAI*, volume 2575 of *LNCS*, pages 206–222. Springer-Verlag, January 2003.

15. Patrice Godefroid, Michael Huth, and Radha Jagadeesan. Abstraction-based model checking using modal transition systems. In *Proceedings of CONCUR'01*, 2001.

16. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In *CAV*, 1997.

17. Michael Huth, Radha Jagadeesan, and David Schmidt. Modal transition systems: A foundation for three-valued program analysis. *LNCS*, 2028:155–169, 2001.

18. Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM (JACM)*, 47(2):312–360, 2000.

19. R.P. Kurshan. *Computer-Aided-Verification of Coordinating Processes*. Princeton University Press, 1994.

20. K.G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210, 1988.

21. Woohyuk Lee, Abelardo Pardo, Jae-Young Jang, Gary D. Hachtel, and Fabio Somenzi. Tearing based automatic abstraction for CTL model checking. In *ICCAD*, pages 76–81, 1996.

22. Martin Leucker. Model checking games for the alternation free mu-calculus and alternating automata. In *Conf. on Logic for Programming and Automated Reasoning (LPAR)*, 1999.

23. Jorn Lind-Nielsen and Henrik Reif Andersen. Stepwise CTL model checking of state/event systems. In *Computer Aided Verification*, pages 316–327, 1999.

24. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 1995.

25. Kedar S. Namjoshi. Certifying model checkers. In *CAV*, volume 2102 of *LNCS*, 2001.

26. Kedar S. Namjoshi and Robert P. Kurshan. Syntactic program transformations for automatic abstraction. In *CAV*, volume 1855 of *LNCS*, pages 435–449. Springer, 2000.

27. Abelardo Pardo and Gary D. Hachtel. Automatic abstraction techniques for propositional mu-calculus model checking. In *Computer Aided Verification*, pages 12–23, 1997.

28. Abelardo Pardo and Gary D. Hachtel. Incremental CTL model checking using BDD subsetting. In *Design Automation Conference (DAC)*, pages 457–462, 1998.

29. Doron Peled and Lenore Zuck. From model checking to a temporal proof. In *SPIN*, 2001.

30. H. Saidi and N. Shankar. Abstract and model check while you prove. In *CAV*, 1999.

31. Colin Stirling. *Modal and Temporal Properties of Processes*. Springer, 2001.

32. Li Tan and Rance Cleaveland. Evidence-based model checking. In *CAV*, 2002.