A Game-Theoretic Approach to Branching Time Abstract-Check-Refine Process

Yi Wang
The University of Tokyo
Meguro-ku, Tokyo, Japan
wangyi@graco.c.u-tokyo.ac.jp

Tetsuo Tamai
The University of Tokyo
Meguro-ku, Tokyo, Japan
tamai@graco.c.u-tokyo.ac.jp

Abstract

Since the complexity of software systems continues to grow, most engineers face two serious problems: the state space explosion problem and the problem of how to debug systems. In this paper, we propose a game-theoretic approach to full branching time model checking on three-valued semantics. The three-valued models and logics provide successful abstraction that overcomes the state space explosion problem. The game style model checking that generates counterexamples can guide refinement or identify validated formulas, which solves the system debugging problem. Furthermore, output of our game style method will give significant information to engineers in detecting where errors have occurred and what the causes of the errors are.

1 Introduction

Model checking is a major technique for verifying finite state systems [5]. The procedure normally uses an exhaustive search of the state space of the system to determine whether some specification is satisfied or not. Given sufficient resources, the procedure will always terminate with a yes/no answer. Since the size of a given system (concrete model) is usually very large, the state explosion problem may occur in model checking such a model. Abstraction as an indispensable means to reduce the state space makes model checking feasible [4, 5]. The traditional abstraction method uses two-valued semantics. It brings about two kinds of situations: under-approximation and over-approximation. In under-approximation, the abstract model exhibits behavior that exists in the concrete model but may miss some of its behavior. On the other hand, over-approximation may bring in additional behavior that does not exist in the concrete model. Unfortunately both of these two-valued abstractions have the unsoundness problem. That is for every existential property satisfied by an under-approximate model also holds in the concrete model but universal properties (for example safety) do not necessarily hold, and for every universal property satisfied by an over-approximate model also holds in the concrete model but existential properties (for example liveness) do not necessarily hold. Such unsoundness makes abstraction less useful and the state space explosion problem still remains. In the past ten years, three-valued models and logics have been studied. The main benefit of this approach is that both universal and existential properties are guaranteed to be sound. The three-valued semantics evaluates a formula to either *true, false* or an indefinite value as the third value. In Kleene's strongest regular three-valued propositional logic, the third value is understood as *unknown* that means it can take either *true* or *false*. The three-valued models, or modal transition systems, contain *may*-transitions which over-approximate transitions of the concrete model, and *must*-transitions which under-approximate transitions of the concrete model. To ensure logical consistency, truth of universal formulas is then examined over *may*-transitions, whereas truth of existential formulas is examined over *must*-transitions. We follow this approach.

Full branching time logic CTL* as an expressive fragment [6, 2, 5] of μ-calculus can describe properties in computation trees. There has been much work on model checking for sublogic of CTL* such as LTL and CTL [11, 3], but little on CTL*. This paper introduces ideas for full branching time temporal logic CTL*.

There are several approaches to studying computations: computational model approach, algebraic approach, logical approach and game-theoretic approach. In the field of model checking, one uses the logical approach to capture temporal ordering of events, the algebraic approach to model examined systems, and the computational model approach and the game-theoretic approach to do model checking. A typical model checking approach is the automata-theoretic approach. Kupferman, Vardi and Wolper have shown how to solve the model checking problem for branching time by using alternating automata [9]. In their approach the model checking problem is reduced to a non-emptiness checking problem of the alternating automaton composed as a product between a *Kripke structure* and an automaton expressing the interesting property. The game-theoretic approach to model checking can also be viewed as simulating alternating automata [10, 12]. Their winning conditions correspond to a special Rabin acceptance condition of the automata approach. In contrast to the automata-theoretic model checking approach it is not necessary to compose automata for the properties.

Related Work. Martin Lange and Colin Stirling proposed Model checking games for CTL* in [10]. They described a two-player CTL* *focus game* for *Kripke structure* on Boolean semantics. We propose a generalization of the two-player game for *Modal Transition System* on three-valued semantics. There were many papers of three-valued abstraction that proposed by Godefroid, Jagadeesan and Bruns [1, 7, 8]. They showed advantages of three-valued models. In this paper, we not only discuss advantages of three-valued models but also analyze the returning information for debugging, proving or refining such models. Sharon Shoham and Orna Grumberg proposed muti-valued games for μ-calculus [12]. We investigate games for CTL* which is the most expressive fragment of μ-calculus. To sum up, the contribution of this paper are: (1) new game-based approach to three-valued model checking, (2) new game-based algorithm showing winning strategy of each player for solving the game, (3) new analysis based on *focus game* for debugging, proving or refining abstract models.

2 Preliminaries

We introduce key notions behind the framework of abstraction and model checking. Let AP be a finite set of atomic propositions. We define that an atomic proposition p is in AP if and only if its negation $\neg p$ is in AP. In the rest of this paper we suppose that all models, both abstract and concrete, share the set AP.

The full branching time logic CTL* formulas are composed of propositions, negation, Boolean connectives, path quantifiers and temporal operators.

Definition 1 (Syntax of CTL*) *There are two types of formulas in CTL*: state formulas and path formulas. The syntax of state formulas is given by the following rules:*
If p is an atomic proposition, then p is a state formula.
If ψ_1 and ψ_2 are state formulas, then $\neg\psi_1$, $\psi_1 \vee \psi_2$ and $\psi_1 \wedge \psi_2$ are state formulas.
If ψ is a path formula, then $\mathrm{E}\psi$ and $\mathrm{A}\psi$ are state formulas.
If ψ is a state formula, then ψ is also a path formula.
If ψ_1, ψ_2 are path formulas then $\neg\psi_1, \psi_1 \vee \psi_2, \psi_1 \wedge \psi_2, \mathrm{X}\psi_1, \mathrm{F}\psi_1, \mathrm{G}\psi_1, \psi_1\mathrm{U}\psi_2$, and $\psi_1\mathrm{R}\psi_2$ are path formulas.

The F, G temporal operators can be replaced with U, R operators by rules: $\mathrm{F}\psi = \textbf{true}\ \mathrm{U}\psi$, $\mathrm{G}\psi = \textbf{false}\ \mathrm{R}\psi$. The set of subformulas $Sub(\varphi)$ for a given φ is defined in the usual way, except that
- $Sub(\varphi\mathrm{U}\psi) := \{\varphi\mathrm{U}\psi, X(\varphi\mathrm{U}\psi), \varphi \wedge X(\varphi\mathrm{U}\psi), \psi \vee (\varphi \wedge X(\varphi\mathrm{U}\psi))\} \cup Sub(\varphi) \cup Sub(\psi)$,
- $Sub(\varphi\mathrm{R}\psi) := \{\varphi\mathrm{R}\psi, X(\varphi\mathrm{R}\psi), \varphi \vee X(\varphi\mathrm{R}\psi), \psi \wedge (\varphi \vee X(\varphi\mathrm{R}\psi))\} \cup Sub(\varphi) \cup Sub(\psi)$.

We consider that every CTL* formula begins with a path quantifier "A" to ensure that it is a state formula. The following semantics of CTL* shows that this is not a restriction because of the equivalence $Q_1Q_2\varphi = Q_2\varphi$ for $Q_1, Q_2 \in \{\mathrm{A}, \mathrm{E}\}$.

Definition 2 (Semantics of CTL*) *Suppose that π^i denotes the suffix of π starting at s_i. Let ψ be CTL* formula and M be a model. If ψ is a state formula, the notation $M, s \models \psi$ means that ψ holds at state s*

in model M. Similarly, if ψ is a path formula, $M,\pi \models \psi$ means that ψ holds along path π in model M. The relation $\models$ is defined inductively as follows (assuming that ψ_1 and ψ_2 are state formulas and ψ_1' and ψ_2' are path formulas):

$M,s \models p \iff p \in L(s)$.
$M,s \models \psi_1 \vee \psi_2 \iff M,s \models \psi_1$ or $M,s \models \psi_2$.
$M,s \models \mathrm{E}\psi_1' \iff$ there is a path π from s such that $M,\pi \models \psi_1'$.
$M,\pi \models \psi_1 \iff s$ is the first state of π and $M,s \models \psi_1$.
$M,\pi \models \psi_1' \wedge \psi_2' \iff M,\pi \models \psi_1'$ and $M,\pi \models \psi_2'$.
$M,\pi \models \mathrm{F}\psi_1' \iff$ there exists a $k \geq 0$ such that $M,\pi^k \models \psi_1'$.
$M,\pi \models \psi_1'\mathrm{R}\psi_2' \iff$ for all $j \geq 0$, if for every $i < j$ $M,\pi^i \not\models \psi_1'$ then $M,\pi^j \models \psi_2'$.
$M,s \models \neg\psi_1 \iff M,s \not\models \psi_1$.
$M,s \models \psi_1 \wedge \psi_2 \iff M,s \models \psi_1$ and $M,s \models \psi_2$.
$M,s \models \mathrm{A}\psi_1' \iff$ for every path π starting from s, $M,\pi \models \psi_1'$.
$M,\pi \models \neg\psi_1' \iff M,\pi \not\models \psi_1'$.
$M,\pi \models \psi_1' \vee \psi_2' \iff M,\pi \models \psi_1'$ or $M,\pi \models \psi_2'$.
$M,\pi \models \mathrm{X}\psi_1' \iff M,\pi^1 \models \psi_1'$.
$M,\pi \models \mathrm{G}\psi_1' \iff$ for all $i \geq 0$, $M,\pi^i \models \psi_1'$.
$M,\pi \models \psi_1'\mathrm{U}\psi_2' \iff$ there exists a $k \geq 0$ such that $M,\pi^k \models \psi_2'$ and for all $0 \leq j < k$, $M,\pi^j \models \psi_1'$.

Consider that every concrete model is given as a *Kripke structure* (KS for short) over AP, denoted by M_c. A KS is four tuple $\langle S_c, S_c^0, R_c, L_c\rangle$ where S_c is a finite set of states. $S_c^0 \subseteq S_c$ is the set of initial states. $R_c \subseteq S_c \times S_c$ is a transition relation that must be total, that is, for every state $s_c \in S_c$ there is a state $s' \in S_c$ such that $s_c \rightarrow s_c' \in R_c$. $L_c : S_c \rightarrow 2^{AP}$ is a labeling function such that for every state s and every $p \in AP$, $p \in L_c(s)$ iff $\neg p \notin L_c(s)$. $[M_c \models \varphi] = \mathrm{tt}$ $(= \mathrm{ff})$ or $M_c \models \varphi$ $(M_c \not\models \varphi)$ means that M_c satisfies (refutes) the CTL* formula φ.

An abstraction (S_a, γ) for S_c consists of a finite set of abstract states S_a and a total concretization function $\gamma : S_a \rightarrow 2^{S_c}$ that maps each abstract state to the (nonempty) set of concrete states it represents. The function $\alpha : 2^{S_c} \rightarrow S_a$ as the inverse of γ is said to be abstraction function. An abstract model M_a is said to be on two-valued semantics if it is a KS model. An abstract model is said to be on three-valued semantics if it is a *Modal Transition System* (MTS) model [1, 7, 8]. MTSs contain two types of transitions: *may*-transitions and *must*-transitions.

Definition 3 *A Modal Transition System M_a over AP is a tuple $\langle S_a, S_a^0, R_{may}, R_{must}, L_a\rangle$, where S_a is a nonempty finite set of states, $S_a^0 \subseteq S_a$ is the set of initial states, $R_{may} \subseteq S_a \times S_a$ and $R_{must} \subseteq S_a \times S_a$ are transition relations such that $R_{must} \subseteq R_{may}$. $L_a : S_a \rightarrow 2^{AP}$.*

R_{may} is the set of all possible transitions and R_{must} is the set of all inevitable transitions. Note that $R_{must} \subseteq R_{may}$, because all inevitable transitions are possible transitions. Consider a concrete KS M_c and an abstract MTS M_a of M_c. Let (S_a, γ) be the three-valued abstraction between M_c and M_a. Labelings in each state in S_a are constructed by the following rules:

$$\frac{p \in L_c(s) \wedge \neg p \in L_c(s')}{p \notin L_a(s_a)} \qquad \frac{p \in L_c(s) \wedge p \in L_c(s')}{p \in L_a(s_a)} \qquad \frac{\neg p \in L_c(s) \wedge \neg p \in L_c(s')}{\neg p \in L_a(s_a)}$$

where $s, s' \in \gamma(s_a)$. Note that it is possible that neither p nor $\neg p$ is in $L_a(s_a)$ though either p or $\neg p$ must be in $L_c(s_c)$. After state abstraction, transitions in M_a can be constructed by using the following rules:

$$\frac{\exists s_1 \in \gamma(s_a), \exists s_2 \in \gamma(s_a') : s_1 \rightarrow s_2}{s_a \overset{may}{\rightarrow} s_a'} \qquad \frac{\forall s_1 \in \gamma(s_a), \exists s_2 \in \gamma(s_a') : s_1 \rightarrow s_2}{s_a \overset{must}{\rightarrow} s_a'}$$

Other constructions of abstract models are based on *Galois connections*, which can be found in [7]. The three-valued semantics of CTL* over MTSs, denoted $[M \models^3 \varphi]$, preserving both satisfaction and refutation from the abstract model M_a to the concrete model M_c. However, a new truth value (the third value *unknown*), denoted by $\perp$, is introduced meaning that the truth value over the concrete model is not

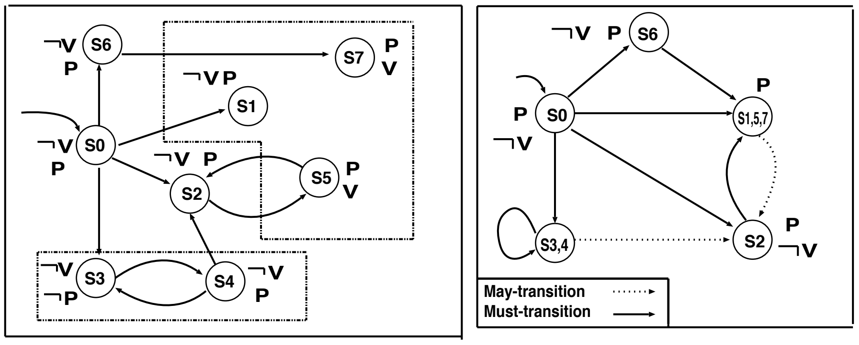

Figure 1: Example of Three-Valued Abstraction

known that can either be the truth value *true* or *false*.
Example 4 Let $AP = \{p, \neg p, v, \neg v\}$. Figure 1 shows a concrete KS model M_c (left) and its abstract MTS model M_a (right). Let (S_a, γ) be an (three-valued) abstraction, where $S_a = \{s_0, s_2, s_{3,4}, s_{1,5,7}, s_6\}$ and $\gamma(s_0) = \{s_0\}$; $\gamma(s_2) = \{s_2\}$; $\gamma(s_{3,4}) = \{s_3, s_4\}$; $\gamma(s_{1,5,7}) = \{s_1, s_5, s_7\}$; $\gamma(s_6) = \{s_6\}$.

3 Generalization of Focus Game

Model checking games for CTL* over two-valued models are two-player games, called *focus game*, which is proposed by Lange and Stirling [10]. There are two players in the *focus game*: the first player $\forall$ and the second player $\exists$. $\forall$'s task is to show that a formula is unsatisfied, while $\exists$'s task is to show the converse. In the two-player *focus game*, the set of *configurations* for a model (system) M and a formula φ, written in CTL*, is $conf(M, \varphi) = \{\forall, \exists, \bot\} \times S \times Sub(\varphi) \times 2^{Sub(\varphi)}$. A configuration is written $p, s, [\psi], \Phi$ where p is a player called the *path player*, $s \in S$, $\psi \in Sub(\varphi)$ and $\Phi \subseteq Sub(\varphi)$. Here ψ is said to be *in focus* and Φ are called side formulas. If p denotes a player then $\bar{p}$ denotes the other one in any round. A *play* between player p and $\bar{p}$ is a sequence of configurations. There are eighteen rules of the form

$$\frac{p, s, [\varphi], \Phi}{p', s, [\varphi'], \Phi'} p''$$

for transforming configurations, where $p, p', p'' \in \{\forall, \exists, \bot\}$ denote players. We follow these definitions and propose a three-player game for evaluating a CTL* formula φ on an abstract MTS model M_a with respect to three-valued semantics. We generalize the game by inserting a new player (the third player) $\bot$ and setting the game played in two rounds. Without loss of generality, we match $\forall$ *vs.* $\bot$ in the first round and $\exists$ *vs.* $\bot$ in the second round. We define that $\forall$ wins the game if $\forall$ wins the first round; $\exists$ wins the game if $\exists$ wins the second round; $\bot$ wins the game if $\bot$ wins both rounds. At each configuration the set of side formulas together with the formula in focus can be understood as a disjunction(resp. conjunction) of formulas in case the path player is $\forall$(resp. $\bot$) in the first round, $\bot$(resp. $\exists$) in the second round.

A play for model M with starting state s and a formula begins with the configuration $\forall, s, [\varphi]$ in the first round and with the configuration $\bot, s, [\varphi]$ in the second round. There are eighteen rules in the two-player focus game.

Path-chosen rules : Discarding rules :

$$(1)\frac{p, s, [A\varphi], \Phi}{\forall, s, [\varphi]} \qquad (2)\frac{p, s, [E\varphi], \Phi}{\exists, s, [\varphi]} \qquad (3)\frac{p, s, [\varphi], Q\psi, \Phi}{p, s, [\varphi], \Phi} \bar{p} \qquad (4)\frac{p, s, [\varphi], q, \Phi}{p, s, [\varphi], \Phi} \bar{p}$$

Boolean-connection rules :

$$(5)\frac{\forall,s,[\varphi_0\wedge\varphi_1],\Phi}{\forall,s,[\varphi_i],\Phi}\forall \quad (6)\frac{\forall,s,[\varphi_0\vee\varphi_1],\Phi}{\forall,s,[\varphi_i],\varphi_{1-i},\Phi}\exists \quad (7)\frac{\exists,s,[\varphi_0\vee\varphi_1],\Phi}{\exists,s,[\varphi_i],\Phi}\exists \quad (8)\frac{\exists,s,[\varphi_0\wedge\varphi_1],\Phi}{\exists,s,[\varphi_i],\varphi_{1-i},\Phi}\forall$$

Unfolding rules :

$$(9)\frac{p,s,[\varphi\mathrm{U}\psi],\Phi}{p,s,[\psi\vee(\varphi\wedge\mathrm{X}(\varphi\mathrm{U}\psi))],\Phi} \qquad (10)\frac{p,s,[\varphi\mathrm{R}\psi],\Phi}{p,s,[\psi\wedge(\varphi\vee\mathrm{X}(\varphi\mathrm{R}\psi))],\Phi}$$

Progress rules :

$$(11)\frac{\forall,s,[\mathrm{X}\psi],\varphi_0\wedge\varphi_1,\Phi}{\forall,s,[\mathrm{X}\psi],\varphi_i,\Phi}\forall(12)\frac{\forall,s,[\mathrm{X}\psi],\varphi_0\vee\varphi_1,\Phi}{\forall,s,[\mathrm{X}\psi],\varphi_0,\varphi_1,\Phi} \quad (13)\frac{\exists,s,[\mathrm{X}\psi],\varphi_0\vee\varphi_1,\Phi}{\exists,s,[\mathrm{X}\psi],\varphi_i,\Phi}\exists(14)\frac{\exists,s,[\mathrm{X}\psi],\varphi_0\wedge\varphi_1,\Phi}{\exists,s,[\mathrm{X}\psi],\varphi_0,\varphi_1,\Phi}$$

$$(15)\frac{p,s,[\mathrm{X}\chi],\varphi\mathrm{U}\psi,\Phi}{p,s,[\mathrm{X}\chi],\psi\vee(\varphi\wedge\mathrm{X}(\varphi\mathrm{U}\psi)),\Phi} \qquad (16)\frac{p,s,[\mathrm{X}\chi],\varphi\mathrm{R}\psi,\Phi}{p,s,[\mathrm{X}\chi],\psi\wedge(\varphi\vee\mathrm{X}(\varphi\mathrm{R}\psi)),\Phi}$$

To apply our three-player game, we restrict moves for different players. Since transitions may take place only in configurations with subformulas of the form $X\psi$, it is the only case where the rule (17) need to be applied to *may*-transitions and *must*-transitions.
$\forall$ and $\exists$ move on *must*-transitions: $\perp$ moves on *may* transitions:

$$(17a)\frac{p,s,[\mathrm{X}\varphi_0],\mathrm{X}\varphi_1,\cdots,\mathrm{X}\varphi_k}{p,t,[\varphi_0],\varphi_1,\cdots,\varphi_k}p\in\{\forall,\exists\},s\overset{must}{\rightarrow}t \qquad (17b)\frac{\perp,s,[\mathrm{X}\varphi_0],\mathrm{X}\varphi_1,\cdots,\mathrm{X}\varphi_k}{\perp,t,[\varphi_0],\varphi_1,\cdots,\varphi_k}\perp,s\overset{may}{\rightarrow}t$$

The special rule (Change focus)

$$(18)\frac{p,s,[\varphi],\psi,\Phi}{p,s,[\psi],\varphi,\Phi}\bar{p}$$

A *move* in a play consists of two steps. First the path player and the focus determines which of the rules (1) – (17a) or (1) – (17b) apply, and hence which player makes the next choice. After that the path player's opponent has the chance to reset the focus by using rule (18). A play is finished after a full move if it has reached one of the following configurations (finish conditions).
1. $p,s,[q],\Phi$.
2. $C=\exists,s,[\varphi\mathrm{U}\psi],\Phi$ after the play already went through C and $\forall$ never applied (18) in between.
3. $C=\forall,s,[\varphi\mathrm{R}\psi],\Phi$ after the play already went through C and $\exists$ never applied (18) in between.
4. $p,s,[\varphi],\Phi$ for the second time possibly using rule (18) in between.
6. $\forall,s,[X\psi],\mathrm{X}\varphi_1,\cdots,\mathrm{X}\varphi_k$ and the rule (17a) can not be applied.
7. $\exists,s,[X\psi],\mathrm{X}\varphi_1,\cdots,\mathrm{X}\varphi_k$ and the rule (17a) can not be applied.
The winning criteria for three-player game are:
If the rule (17b) has been applied in a play and the play ends with one of the above finish conditions then $\perp$ wins, else
In the first round ($\forall$ **vs.** $\perp$)
1. When a play ends with the first finish condition, $\forall$ wins if $\neg q\in L(s)$, otherwise $\perp$ wins.
2. When a play ends with the second finish condition, $\forall$ wins.
3. When a play ends with the third finish condition, $\perp$ wins.
4. When a play ends with the fourth finish condition, the path player p wins if the second or the third finish condition does not apply.
5. Whenever a play ends with the fifth or the sixth finish condition, $\perp$ wins.
In the second round ($\exists$ **vs.** $\perp$)

1. When a play ends with the first finish condition, $\exists$ wins if $q \in L(s)$, otherwise $\perp$ wins.
2. When a play ends with the second finish condition, $\perp$ wins.
3. When a play ends with the third finish condition, $\exists$ wins.
4. When a play ends with the fourth finish condition, the path player p wins if the second or the third finish condition does not apply.
5. Whenever a play ends with the fifth or the sixth finish condition, $\perp$ wins.

The second round of the game is not always played. It is played if $\perp$ wins the first round, else the game is over and $\forall$ wins the game. Note that the game on three-valued semantics is an unfair game. Players $\forall$ and $\exists$ cannot move on all *may*-transitions whereas $\perp$ can move.

Let $\Gamma_{CTL^*}(M,s,\varphi)$ be a game over M for a CTL* formula φ. A game $\Gamma_{CTL^*}(M,s,\varphi)$ can be described as trees of all possible plays.

Definition 5 *A (game) tree is said to be a winning tree for player p, if p wins every branch (play) in it.*

We say that the player *p wins* or has a *winning strategy* for $\Gamma_{CTL^*}(M,s,\varphi)$ if p can force every play into a configuration that makes p win the play. A player p wins a round if p wins all possible plays in that round.

Example 6 Let $\varphi = AX(p) \vee EF(v)$ be a property that we want to check. Let M_a be the abstract model given in figure 1. We show that $\perp$ wins the game ($\perp$ wins both rounds) with the following winning trees.

The first round.

$$\begin{array}{cc}
\multicolumn{2}{c}{\forall, s_0, [\varphi]} \\ \hline
\forall, s_0, [AX(p)], EF(v) & \forall, s_0, [EF(v)], AX(p) \\ \hline
\forall, s_0, [AX(p)] & \perp, s_0, [F(v)] \\ \hline
\forall, s_0, [X(p)] & \perp, s_0, [v \vee XF(v)] \\ \hline
\forall, s_{3,4}, [p] & \perp, s_0, [XF(v)] \\ \hline
 & \perp, s_{1,5,7}, [F(v)] \\ \hline
 & \perp, s_{1,5,7}, [v \vee XF(v)] \\ \hline
 & \perp, s_{1,5,7}, [v]
\end{array}$$

The second round.

$$\begin{array}{cc}
\multicolumn{2}{c}{\perp, s_0, [\varphi]} \\ \hline
\perp, s_0, [AX(p)], EF(v) & \perp, s_0, [EF(v)], AX(p) \\ \hline
\perp, s_0, [EF(v)], AX(p) & \exists, s_0, [F(v)] \\ \hline
\cdots & \exists, s_0, [v \vee XF(v)] \\ \hline
 & \exists, s_0, [XF(v)] \\ \hline
 & \exists, s_{1,5,7}, [F(v)] \\ \hline
 & \exists, s_{1,5,7}, [v \vee XF(v)] \\ \hline
 & \exists, s_{1,5,7}, [v]
\end{array}$$

4 Game-Based Algorithm

In the rest of this paper we assume that M_c, as a Kripke structure, represents a given concrete model and M_a, a Modal Transition System, denotes the abstract model of M_c. Let φ be a CTL* formula that represents the property we are interested in and $\Gamma_{CTL^*}(M_a,s,\varphi)$ be the three-player model checking game on the abstract MTS M_a.

We propose a game-based model checking algorithm, called **Mark Configuration**, for solving the game. **Mark Configuration** marks every configuration with one of symbols $\{\forall, \exists, \perp\}$ in each round. Let $C = p, s, [\varphi]$ be the starting configuration. **Mark Configuration** runs recursively and finally marks the starting configuration C with one of the three symbols.

Mark Configuration (the 1st round)

```
1.    BEGIN Mark($C$)
2.    INITIAL : $history = \emptyset$ ;
3.1   IF $\varphi = A\psi$ THEN Mark($\forall, s, [\psi]$)
3.2   ELSE IF $\varphi = E\psi$ THEN Mark($\perp, s, [\psi]$)
3.3   ELSE
3.4     SWITCH($\varphi$)
3.5       CASE 1 – CASE 6 ;
4.    END
```

Mark Configuration (the 2st round)

```
1.    BEGIN Mark($C$)
2.    INITIAL : $history = \emptyset$ ;
3.1   IF $\varphi = A\psi$ THEN Mark($\perp, s, [\psi]$)
3.2   ELSE IF $\varphi = E\psi$ THEN Mark($\exists, s, [\psi]$)
3.3   ELSE
3.4     SWITCH($\varphi$)
3.5       CASE 1 – CASE 6 ;
4.    END
```

From the syntax of CTL*, six types of formula φ can be considered when it is neither started by A nor E. That is $\varphi = q \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2 \mid X\psi \mid \psi_1 U \psi_2 \mid \psi_1 R \psi$. Each of them corresponds to one case in this algorithm. When the focus formula φ is q, $\psi_1 \wedge \psi_2$ or $\psi_1 \vee \psi_2$, the configuration C's mark is decided by its children's marks. Let C_1', C_2' be configurations with subformula ψ_1, ψ_2, respectively. We represents case 1 – 3 as follows.

1. $C = p, s, [q]$ (where $p \in \{\forall, \exists, \perp\}$)
   If p wins in C then return p else return $\bar{p}$;

2. $C = p, s, [\psi_1 \wedge \psi_2]$ (where $p \in \{\forall, \exists, \perp\}$).
   if $\mathbf{Mark}(C_1') = \exists$ and $\mathbf{Mark}(C_2') = \exists$ then return $\exists$;
   if $\mathbf{Mark}(C_1') = \forall$ or $\mathbf{Mark}(C_2') = \forall$ then return $\forall$;
   if $\mathbf{Mark}(C_1') = \exists$ and $\mathbf{Mark}(C_2') = \perp$ or $\mathbf{Mark}(C_1') = \perp$ and $\mathbf{Mark}(C_2') = \exists$ then return $\perp$;

3. $C = p, s, [\psi_1 \vee \psi_2]$ (where $p \in \{\forall, \exists, \perp\}$).
   if $\mathbf{Mark}(C_1') = \forall$ and $\mathbf{Mark}(C_2') = \forall$ then return $\forall$;
   if $\mathbf{Mark}(C_1') = \exists$ or $\mathbf{Mark}(C_2') = \exists$ then return $\exists$;
   if $\mathbf{Mark}(C_1') = \forall$ and $\mathbf{Mark}(C_2') = \perp$ or $\mathbf{Mark}(C_1') = \perp$ and $\mathbf{Mark}(C_2') = \forall$ then return $\perp$;

The function *must-next* (*may-next*) is assumed to calculate all possible successors of a configuration by one move from rules 17a (17b). The configuration C's mark in case $\varphi = X\psi$ is determined not just by its children's marks but also by who the current player is and which the current round is. We distinguish different players in different rounds. The case 4 is as follows.

4a. $C = \forall, s, [X\psi]$ (1st round) or $\perp, s, [X\psi]$ (2nd round).
if there is a $C' \in$ *must-next*(C) and $\mathbf{Mark}(C') = \forall$ then return $\forall$; if for all $C' \in$ *must-next*(C): $\mathbf{Mark}(C') = \exists$ then return $\exists$; if there is $C' \in$ *may-next*(C): $\mathbf{Mark}(C') = \perp$ and for any other $C'' \in$ *may-next*(C): $\mathbf{Mark}(C'') = \exists$ or $\mathbf{Mark}(C'') = \perp$ then return $\perp$;

4b. $C = \perp, s, [X\psi]$ (1st round) or $\exists, s, [X\psi]$ (2nd round).
if for all $C' \in$ *may-next*(C) : $\mathbf{Mark}(C') = \forall$ then return $\forall$; if there is a $C' \in$ *must-next*(C) and $\mathbf{Mark}(C') = \exists$ then return $\exists$; if there is a $C' \in$ *may-next*(C) and $\mathbf{Mark}(C') = \perp$ and for any other $C'' \in$ *may-next*(C): $\mathbf{Mark}(C'') = \forall$ or $\mathbf{Mark}(C'') = \perp$ then return $\perp$;

To determine the configuration C's mark in case $\varphi = \psi_1 \mathrm{U} \psi_2$ and case $\varphi = \psi_1 \mathrm{R} \psi$, first we should look for who is the path player in C. Next we check whether C is the starting configuration of a loop. The variable *history* is used in recording checked configurations in loops on any path. The case 5, 6 are as follows.

5a. $C = p, s, [\psi_1 \mathrm{U} \psi_2]$.
(where $p = \forall$ in 1st round, $p = \perp$ in 2nd round)
if $C \in$ *history* then marks all $C' \in$ *history* with p;
history $:= \emptyset$; return p;
else *history* $:= \{C\} \cup$ *history*;
$\mathbf{Mark}(p, s, [\psi_2 \vee (\psi_1 \wedge X\psi_1 \mathrm{U} \psi_2)])$;

5b. $C = p, s, [\psi_1 \mathrm{U} \psi_2]$
(where $p = \perp$ in 1st round, $p = \exists$ in 2nd round)
if $C \in$ *history* then marks all $C' \in$ *history* with $\bar{p}$;
history $:= \emptyset$; return $\bar{p}$;
else *history* $:= \{C\} \cup$ *history*;
$\mathbf{Mark}(p, s, [\psi_2 \vee (\psi_1 \wedge X\psi_1 \mathrm{U} \psi_2)])$;

6a. $C = p, s, [\psi_1 \mathrm{R} \psi_2]$
(where $p = \forall$ in 1st round, $p = \perp$ in 2nd round)
if $C \in$ *history* then marks all $C' \in$ *history* with $\bar{p}$;
history $:= \emptyset$; return $\bar{p}$;
else *history* $:= \{C\} \cup$ *history*;
$\mathbf{Mark}(p, s, [\psi_2 \wedge (\psi_1 \vee X\psi_1 \mathrm{R} \psi_2)])$;

6b. $C = p, s, [\psi_1 \mathrm{R} \psi_2]$
(where $p = \perp$ in 1st round, $p = \exists$ in 2nd round)
if $C \in$ *history* then marks all $C' \in$ *history* with p;
history $:= \emptyset$; return p;
else *history* $:= \{C\} \cup$ *history*;
$\mathbf{Mark}(p, s, [\psi_2 \wedge (\psi_1 \vee X\psi_1 \mathrm{R} \psi_2)])$;

Proposition 7 (Terminating) *The algorithm* **Mark Configuration** *always terminates.*

Proof. The total number of all configuration can be calculated as $|S| \cdot 2^{|\varphi|}$. Every configuration is marked by **Mark Configuration** only once. □

Proposition 8 *Assume that* **Mark Configuration** *marks all configurations in the graph of* $\Gamma_{CTL^*}(M_a, s, \varphi)$. *The following two statements hold.*
1. Every configuration C *is marked with one of* $\forall$, $\exists$, $\bot$.
2. If a configuration C *is marked with* $p \in \{\forall, \exists, \bot\}$ *then* p *wins the (sub) game of* $\Gamma_{CTL^*}(M_a, s, \varphi)$ *that is started from* C.

Proof. The first statement follows from the fact that every case in **Mark Configuration** marks configurations with one symbol and each case corresponds to one syntax element of the focus formula in C. We show the second statement by the induction on structure of game tree that rooted by C. Without loss of generality, assume that $p \in \{\forall, \exists, \bot\}$ wins a game. When any winning tree of p consists of a single configuration, since the case 1 can be applied, C is marked with the symbol p. When any winning tree of p consists of an infinite sequence, in which the configuration C appears infinitely often, the focus formula of C must be either of the form $\psi_1 \mathrm{U} \psi_2$ or $\psi_1 \mathrm{R} \psi_2$. Suppose that the focus formula is U-formula and C is marked with $\forall$ or $\bot$. According to the case 5a or case 5b, $\forall$ or $\bot$ wins, since ψ_2 never holds. Suppose that the focus formula is R-formula and C is marked with $\exists$ or $\bot$. According to the case 6a or case 6b, $\exists$ or $\bot$ wins, since ψ_2 always holds. For any other structure of the game tree, there is at least a next configuration C' as a child of C that is marked with p. According to the remaining cases in **Mark Configuration**, C's mark is decided by the mark of C' in each corresponding case. By the induction hypothesis, C' is marked with p and it is deduced that p wins the game started by C. □

Proposition 9 *Consider a game has been marked by* **Mark Configuration**. *Let* C_s *be the starting configuration marked with* $\chi (\in \{\forall, \exists, \bot\})$. *For any configuration* C_i, *if* C_i *is marked with* χ *and may-next(*C_i*)* $\neq \emptyset$ *then there is at least one configuration* C_j *such that* $C_j \in$ *may-next(*C_i*) and* C_j *is marked with* χ.

Proof. This follows from the observation that the **Mark Configuration** recursively marks every configuration depending on the marks of its child vertices. The starting configuration is guaranteed to be marked eventually by the exhaustiveness of the search. □

Lemma 10 *The following statements hold.*
1. Every play terminates.
2. Every play has a uniquely determined winner.
3. item For every round of $\Gamma_{CTL^*}(M_a, s, \varphi)$ *one of the players has a winning strategy.*
4. One player wins the game iff the other players do not win.

Proof. Lange and Stirling [10] showed that these four statements hold in two-player games. In $\Gamma_{CTL^*}(M_a, s, \varphi)$, each round can be seen as a two-player game with more constraints. In particular, rules (17a) and (17b) do not introduce new moves to every player. Therefore, these four statements hold in each round, which derives this lemma. □

Theorem 11 (Soundness)
1. $\forall$ *wins* $\Gamma_{CTL^*}(M_a, s, \varphi) \;\Rightarrow\; M_c \not\models \varphi$.
2. $\exists$ *wins* $\Gamma_{CTL^*}(M_a, s, \varphi) \;\Rightarrow\; M_c \models \varphi$.
3. $\bot$ *wins* $\Gamma_{CTL^*}(M_a, s, \varphi) \;\Rightarrow\;$ *both* $M_c \models \varphi$ *and* $M_c \not\models \varphi$ *are possible.*

Proof. We now show the statement 1 (statement 2). Every play in $\forall$'s ($\exists$'s) winning tree terminates either in a loop or at a terminating configuration. If it terminates in a loop then there exists a configuration appearing twice and during the loop $\perp$ ($\exists$) cannot (can) win with the finish condition 2 or 4 (3 or 4). Otherwise it terminates at a configuration in which $\forall$ ($\exists$) wins with the finish condition 1. According to our constraint and winning criteria, the rule (17b) cannot be applied in whole $\forall$'s ($\exists$'s) winning tree. It implies that all plays in winning tree are based on *must*-transition in M_a. Therefore, $\forall$ ($\exists$) can also win on the model M_c. That is $M_c \not\models \varphi$ ($M_c \models \varphi$).
We show the statement 3. $\perp$ has winning trees for both rounds. There must be a play that either terminates at a terminating configuration, or uses the rule (17b) that is based on a transition in $R_{may} - R_{must}$ of M_a. Suppose it terminates at a configuration, denoted by $p, s_a, [q], \Phi$, with finish condition 1. According to abstraction, there are two states $s_c, s'_c \in \gamma(s_a)$ such that $q \in L_c(s_c)$ and $\neg q \in L_c(s'_c)$, or $\neg q \in L_c(s_c)$ and $q \in L_c(s'_c)$. Hence, both $\forall$ wins in M_c and $\exists$ wins in M_c are possible. Information is not sufficient to show $M_c \models \varphi$ or $M_c \not\models \varphi$. □

Lemma 12 *The following two statements hold.*
1. $M_c \models \varphi \;\Rightarrow\; [M_a \models^3 \varphi] = \text{tt}$ or $[M_a \models^3 \varphi] = \perp$.
2. $M_c \not\models \varphi \;\Rightarrow\; [M_a \models^3 \varphi] = \text{ff}$ or $[M_a \models^3 \varphi] = \perp$.

Proof. By the tree-valued abstraction, they are trivial. □

Lemma 13 *The following three statements hold.*
1. $[M_a \models^3 \varphi] = \text{tt} \;\Rightarrow\; \exists$ *wins* $\Gamma_{CTL^*}(M_a, s, \varphi)$.
2. $[M_a \models^3 \varphi] = \text{ff} \;\Rightarrow\; \forall$ *wins* $\Gamma_{CTL^*}(M_a, s, \varphi)$.
3. $[M_a \models^3 \varphi] = \perp \;\Rightarrow\; \perp$ *wins* $\Gamma_{CTL^*}(M_a, s, \varphi)$.

Proof. We show a winning strategy based on **Mark Configuration** for each player $p \in \{\forall, \exists, \perp\}$. Assume that **Mark Configuration** has been applied in the game $\Gamma_{CTL^*}(M_a, s, \varphi)$. Let $p \in \{\forall, \exists, \perp\}$ be a player and C be the current configuration. It does not matter whether p is the path player in C or not. Since $\perp$ can play $\exists$'s role in the first round and can play $\forall$'s role in the second round, we distinguish $\perp$ from other players.
$\forall$'s and $\exists$'s winning strategies: If there is a next configuration C' of C such that C' is marked with symbol p, then select C' by using one of rules 1 – 16, 17a or 18; Else do nothing.
$\perp$'s winning strategy: If there is a next configuration C' of C such that C' is not marked with the mark of $\perp$'s opponent, then select C' by using one of rules 1 – 16, 17b or 18; Else do nothing.
We use Proposition 8, 9 to prove that such strategy is a winning strategy for each corresponding player. Proposition 8 shows that a player p wins the game $\Gamma_{CTL^*}(M_a, s, \varphi)$ if the starting configuration is marked with p. Proposition 9 shows that for any configuration C marked with symbol p, there is a C' such that C' is a next configuration of C and it is also marked with p. By the definition of each player's task, we have $\exists$ wins if $[M_a \models^3 \varphi] = \text{tt}$, $\forall$ wins if $[M_a \models^3 \varphi] = \text{ff}$, $\perp$ wins if $[M_a \models^3 \varphi] = \perp$. □

Theorem 14 (Completeness)
- $M_c \models \varphi \;\Rightarrow\; \exists$ *wins* $\Gamma_{CTL^*}(M_a, s, \varphi)$ *or* $\perp$ *wins.*
- $M_c \not\models \varphi \;\Rightarrow\; \forall$ *wins* $\Gamma_{CTL^*}(M_a, s, \varphi)$ *or* $\perp$ *wins.*

Proof. It directly follows from the Lemma 12 and 13. □

Refinement issues. The main advantage of game-based model checking approach is availability of more precise debugging information on the examined system. Using games is not necessary to create an additional debugger, because the game-based approach annotates each state on the proofs/counterexamples or refinements with a sub-formula of the interesting temporal formula φ that is true/ false or unknown in that state. The annotating sub-formulas being true/false or unknown in the respective states, provide the reason for φ to be true/false or unknown. By analyzing such information we can figure out where errors have occurred and what the causes of the errors are.

Complexity issues. The best currently known complexity for CTL* model checking is in PSPACE time. So is our algorithm. Let all sub-games are started from formula $A\psi$ or $E\psi$. **Mark Configuration** can be applied in every sub-tree in each round. There are at most $|S| \cdot |\varphi|/2$ sub-games. **Mark Configuration** might have to be invoked $|S| \cdot |\varphi|/2$ times. After a sub-game, the space it needs can be released. Thus the algorithm for each round as a two-player game costs PSPACE time. The total complexity is the same class as the complexity of a round.

Conclusions. We presented two problems in the beginning of this paper: the state space explosion problem and the system debugging problem. To overcome both of these problems, we proposed an game-based approach by combining several powerful techniques: abstraction, refinement and three-valued logic. The abstraction on three-valued semantics was used to overcome the first problem. The analysis of the game-based model checking was used to solve the second problem. We also proposed a game-based algorithm for model checking, and proved its termination, soundness and completeness.

References

[1] Glenn Bruns and Patrice Godefroid. Model checking partial state spaces with 3-valued temporal logics. *In Proceedings of the 11th Conference on Computer Aided Verification Lecture Notes in Computer Science*, 1877, July 1999.

[2] Glenn Burns and Patrice Godefroid. Model checking with multi-valued logics. *In Proceedings of the 31st ICALP LNCS*, May 2004.

[3] Edmund M. Clarke and E. Allen Emerson. Synthesis of synchronization skeletons for branching time temporal logic. *Logic of Programs LNCS*, 131, 1981.

[4] Orna Grumberg Edmund M. Clarke and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16, 1994.

[5] Orna Grumberg Edmund M. Clarke, Jr. and Doron A. Peled. *Model Checking*. The MIT Press Cambridge, Massachusetts, London, England, 1999.

[6] Allen E. Emerson and Prasad A. Sistla. Deciding branching time logic. *In Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, March 1984.

[7] Patrice Godefroid and Radaha Jagadeesan. Abstraction-based model checking using modal transition systems. *In Proceedings of CONCUR'(12th International Conference on Concurrency Theory),*, 2001.

[8] Patrice Godefroid and Radaha Jagadeesan. Automatic abstraction using generalized model checking. *In Proceedings of CAV'(14th Conference on Computer Aided Verification),*, July 2002.

[9] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the Association for Computing Machinery*, March 2000.

[10] Martin Lange and Colin Stirling. Model checking games for branching time logics. *Oxford University Press J.Logic Computat.*, 12, 2002.

[11] Amir Pnueli. The temporal logic of programs. *18th IEEE Symposium on the Foundations of Computer Science 46–57*, 1977.

[12] Sharon Shoham and Orna Grumberg. Muti-valued model checking games. *Automated Technology for Verification and Analysis*, 3707, 2005.