Three-Valued Abstractions of Games: Uncertainty, but with Precision

Luca de Alfaro* Patrice Godefroid† Radha Jagadeesan‡

Abstract We present a framework for abstracting two-player turn-based games that preserves any formula of the alternating μ-calculus (AMC). Unlike traditional conservative abstractions which can only prove the existence of winning strategies for only one of the players, our framework is based on 3-valued games, and it can be used to prove and disprove formulas of AMC including arbitrarily nested strategy quantifiers.

Our main contributions are as follows. We define abstract 3-valued games and an alternating refinement relation on these that preserves winning strategies for both players. We provide a logical characterization of the alternating refinement relation. We show that our abstractions are as precise as can be via completeness results. We present AMC formulas that solve 3-valued games with ω-regular objectives, and we show that such games are determined in a 3-valued sense. We also discuss the complexity of model checking arbitrary AMC formulas on 3-valued games and of checking alternating refinement.

1 Introduction

Abstraction is key to extend the scope of formal verification to systems with infinite or very large state spaces, as illustrated by recent work on "software model checking" (e.g., [4, 9, 17]). The relation between the concrete system and an abstraction is usually a simulation relation [31, 28]; this approach enables the verification of universal ($\forall$-CTL*) properties [7]. These abstraction techniques have been extended to games [18, 36] to handle the verification of controllers in open systems. In the context of games, the relation between the concrete and abstract games is usually alternating simulation [3]. While this approach enables the verification of a universal fragment of alternating temporal logic [2], it is unsound for game properties specified using existential path quantifiers and arbitrarily nested path/strategy quantifiers in general.

There are at least four practical motivations to study abstraction frameworks for games that can handle arbitrarily nested strategy quantifiers.

- Compositional verification is one of the motivations for the study of games [13]. In this context, one is interested in the symmetric treatment of all players, and the specification of interesting game-properties requires the full generality of the logic.
- Some applications require nested strategy quantifiers for both players, such as formulas specifying the correctness of security protocols [21, 22].
- Another motivation is to develop a sound theoretical basis for abstraction-based software model-checking tools suitable for both proving and disproving that an open program satisfies a property no matter what its environment does (called *module checking* in [23]). In this context, abstractions are automatically generated from a static analysis of an open program using abstraction techniques such as predicate abstraction [15].
- Even in the traditional framework of conservative abstraction for transition systems, the ability to consider both existential and universal formulas serves as a good foundation for the detection of feasible counter-examples [33][1] and in counter-example refinement [34].

An outline of our approach. In the traditional framework for constructing conservative abstractions using a simulation relation, a transition is present between two abstract states iff there is at least one corresponding transition between related concrete states in the concrete system. This framework is suited only to the verification of universal properties. Modal Transitions Systems (MTS) [25, 24] enable the verification of both universal and existential properties by including two types of transitions: *may-transitions* and *must-transitions*. Typically, a may-transition between abstract states t_1 and t_2 indicates that there is at least one transition from some concrete state related to t_1 to some

*Dept. of Computer Engineering, University of California, Santa Cruz
†Bell Laboratories, Lucent Technologies
‡School of CTI, DePaul University

[1]Though the presentation of the paper itself is not phrased in these logical terms.

concrete state related to t_2, while a must-transition between abstract states t_1 and t_2 indicates that there is a transition from *every* concrete state related to t_1 to some concrete state related to t_2. In an MTS, a universal property is verified by checking the may transitions, and an existential property is verified by checking the must transitions.

When using MTSs to represent abstractions, however, there is an inherent loss of precision in the representation of must-transitions. This loss of precision may prevent the verification of existential properties. To illustrate the problem, consider the following transition system over propositions $\mathtt{p}, \mathtt{q}, \mathtt{r}$, where each state has a superscript with the set of propositions that hold in the state.

$$\begin{tikzcd}
s_1^{\emptyset} \arrow[r] & s_3^{\{\mathtt{p}\}} \arrow[r] & s_5^{\{\mathtt{r}\}} \\
s_2^{\emptyset} \arrow[r] & s_4^{\emptyset} \arrow[ur] &
\end{tikzcd}$$

The CTL formula $\exists\bigcirc([\mathtt{p} \vee \neg\mathtt{q}] \wedge \exists\bigcirc\mathtt{r})$ holds at states s_1, s_2. For simplicity in this discussion, view abstractions as being described by equivalence classes of states, writing $[s_1, s_2]$ for the abstract state which is the equivalence class containing s_1, s_2. Consider abstractions that identify s_1, s_2 and track the predicates $\mathtt{p}, \mathtt{q}, \mathtt{r}$ to prove this formula. The mismatch of propositions in s_3, s_4 and the need to validate $\mathtt{p} \vee \neg\mathtt{q}$ mean that s_3 and s_4 cannot be in the same equivalence class. Thus, in standard MTSs, there is no must-transition from $[s_1, s_2]$ in the abstraction since there is no single equivalence class that can be reached from both s_1 and s_2. On the other hand, if we allow must-transitions to have *sets* as destinations, the abstraction can include a must-transition from $[s_1, s_2]$ to the *set* $\{[s_3], [s_4]\}$. We view these sets conjunctively, i.e., to use a must transition from $[s_1, s_2]$ to $\{[s_3], [s_4]\}$ as a witness to existentially achieve a formula, both $[s_3], [s_4]$ have to satisfy the formula. The need for must-transitions whose targets can be a set of states was first pointed out in [26], and it has been independently advocated in [35, 12].

Motivated by the above discussion, we define in this paper an expressive variant of MTSs, which we call *abstract transition structures*, and study its properties. We show that for any given relation between concrete and abstract states, our abstract transition structures are as precise as possible, among *all* possible abstractions, regardless of their representation. This precision leads to a *compositional framework for the construction of abstractions*. It also enables us to show for a logic of branching-time safety properties that, if such a branching-time safety property holds at a state of a possibly-infinite concrete system, then there is a *finite* abstract system that can prove this.

We then generalize these basic ideas in a natural way to two-player turn-based games[2]. We define an alternating refinement relation between states of an abstract game. Must and may-transitions for the two players give rise to *must-strategies,* that are guaranteed to exist in all refinements, and *may-strategies,* that may exist in a refinement. In contrast to alternating simulation, our definition is *symmetric* between the players, and focuses on the increase in must-transitions and decrease in may-transitions. However, it shares some of the structural features of alternating simulation — e.g., if a state s' is refined by s, every must-strategy (for any subset of players) is preserved from s' to s, and every may-strategy (for any subset of players) is preserved from s to s'. Thus, our definition of alternating refinement preserves winning must-strategies for *all* players.

The determinacy of standard two-player games is replaced by three-valued determinacy. We show that for any objective ϕ there are three possibilities: (a) player 1 can achieve ϕ with a must-strategy, (b) player 2 can achieve $\neg\phi$ with a must-strategy, or (c) both players may be able to achieve their respective, and complementary, objectives using may-strategies. The last unusual case indicates that the abstract system under consideration can potentially be refined in two different ways (wrt. the control-objective ϕ) — one in which player 1 achieves ϕ and a second in which player 2 achieves $\neg\phi$.

As a computational logic for encoding properties and algorithms, we provide a 3-valued (true, false, unknown) semantics for alternating μ-calculus (AMC). We show that all AMC formulas are preserved under refinement. We also provide a logical characterization of the alternating refinement relation in terms of a 3-valued interpretation of AMC formulas. This framework permits the use of existing abstraction techniques for (closed) transition systems [10, 11] in the context of abstraction of games. Our framework is flexible enough to show that *any* classical abstract interpretation of data values extends to an alternating refinement of abstract games that preserves all AMC-formulas.

We also show that, given an LTL or ω-regular objective, the sets of states where a player has must or may-strategies can be computed using AMC formulas. These formulas can thus be used to solve verification and control problems on abstract transition structures and abstract games. We finally discuss the complexity of 3-valued model-checking of AMC formulas on abstract transition structures and abstract games, as well as of checking alternating refinement.

2. Abstract Game Structures

Given a set S, we write $\mathcal{P}(S)$ for the powerset of S. We consider a fixed set Σ of propositions. We consider

[2] Our ideas work for multi-player turn-based games, but are not directly applicable to concurrent games.

3-valued logic, where the three truth values $\mathtt{T}, \mathtt{F}, \bot$ are ordered according to the information ordering: $\bot < \mathtt{T}$ and $\bot < \mathtt{F}$, while $\mathtt{T}$ and $\mathtt{F}$ are incomparable. The boolean operations are defined as follows: $x \vee y$ is true if either is true, false if both are false and $\bot$ otherwise. Negation interchanges true and false, but leaves $\bot$ untouched. We will use $\sqcap, \sqcup$ for the greatest lower bound and the least upper bounds with respect to this information ordering. For two functions $f, g : \Sigma \mapsto \{\mathtt{T}, \mathtt{F}, \bot\}$, we write $f \leq g$ (resp. $f < g$) if $f(q) \leq g(q)$ (resp. $f(q) < g(q)$) for all $q \in \Sigma$. A two-player, turn-based abstract game structure is defined as follows.

Definition 2.1 (abstract game structure). A two-player turn-based *abstract game structure* $\mathcal{G} = \langle S, \lambda, \delta_{\mathtt{may}}, \delta_{\mathtt{must}}, \mathcal{I} \rangle$ consists of:

- A (possibly infinite) state space S.
- $\lambda : S \mapsto \{1, 2\}$ indicates which player is to move at a state. We write $\overline{1} = 2, \overline{2} = 1$.
- A transition function $\delta_{\mathtt{may}} : S \mapsto \mathcal{P}(S)$, which associates with every state $s \in S$ a nonempty set of possible *may-destination states* $\delta_{\mathtt{may}}(s) \subseteq S$.
- A transition function $\delta_{\mathtt{must}} : S \mapsto \mathcal{P}(\mathcal{P}(S))$, which associates with every state $s \in S$ a nonempty set of possible *must-destination state-sets* $\delta_{\mathtt{must}}(s) \subseteq 2^S$ that satisfies the following constraint:

$$(\forall s \in S)\ (\forall U \in \delta_{\mathtt{must}}(s))\ U \cap \delta_{\mathtt{may}}(s) \neq \emptyset$$

- Two predicate valuations $\mathcal{I}_{\mathtt{may}}, \mathcal{I}_{\mathtt{must}} : \Sigma \mapsto \mathcal{P}(S)$, associating with each propositional symbol q the set of states $\mathcal{I}_{\mathtt{may}}(q)$ where q *may* hold, and the set of states $\mathcal{I}_{\mathtt{must}}(q)$ where q *must hold*. They satisfy $\mathcal{I}_{\mathtt{must}}(q) \subseteq \mathcal{I}_{\mathtt{may}}(q)$. ∎

An *abstract transition structure* is the special case of abstract game structure where $\lambda(s) = 1$ for all $s \in S$. Given a set $A \subseteq \{1, 2\}$ of players, we define $\neg A = \{1, 2\} \setminus A$.

An abstract game structure is *finite-branching* if for all $s \in S$, the sets $\delta_{\mathtt{may}}(s)$, $\delta_{\mathtt{must}}(s)$, and all the sets in $\delta_{\mathtt{must}}(s)$, are finite. An abstract game structure is *concrete* if for all states $s \in S$, we have $\delta_{\mathtt{must}}(s) = \{\{t\} \mid t \in \delta_{\mathtt{may}}(s)\}$, and if for all $q \in \Sigma$, we have $\mathcal{I}_{\mathtt{may}}(q) = \mathcal{I}_{\mathtt{must}}(q)$.

In contrast to other existing models with 3-valued propositions and transition functions [11, 5, 19], this definition follows [26] in the type of the must-transition function, which associates to a state a set of successor *state-sets,* rather than states. In the context of abstract transition systems, this definition has been independently explored in [35] and with fairness considerations in another paper in this volume [12].

May and must-transitions can be understood as follows. In a may-transition from $s \in S$, we can choose the destination state $t \in \delta_{\mathtt{may}}(s)$: hence, to achieve $T \subseteq S$ at the next state, it suffices that $\delta_{\mathtt{may}}(s) \cap T \neq \emptyset$. Must-transitions can be interpreted as a game between the system and nondeterminism: the system chooses $U \in \delta_{\mathtt{must}}(s)$, and the nondeterminism chooses $t \in U$. To achieve $T \subseteq S$ at the next state, there must be $U \in \delta_{\mathtt{must}}(s)$ with $U \subseteq T$.

In view of this interpretation, for each state s we need to represent only the elements of $\delta_{\mathtt{must}}(s)$ that are minimal with respect to set inclusion. Precisely, if $T, T' \in \delta_{\mathtt{must}}(s)$ with $T \subset T'$, then we can remove T' from $\delta_{\mathtt{must}}(s)$ without affecting the ability of either player to win the game: in fact, the player that moves at s will always prefer move T to T'. We will make this remark precise below, showing that removing T' preserves game similarity.

The condition on the may/must-transition relations can be viewed as an analogue of the consistency condition of modal transition systems. To see this, change the type of the may-transition function to $\delta^u_{\mathtt{may}}(s) \subseteq \mathcal{P}(S)$, let $\delta^u_{\mathtt{may}}(s) = \{\{t\} \mid t \in \delta_{\mathtt{may}}(s)\}$ and $\delta^u_{\mathtt{must}}(s) = \delta_{\mathtt{must}}(s)$, and "upward close" both transition functions, i.e., for all $s \in S$, all $\gamma \in \{\mathtt{may}, \mathtt{must}\}$ and all $T \in \delta^u_\gamma(s)$, if $T \subseteq T' \subseteq S$, then add also T' to $\delta^u_\gamma(s)$. In such a setting, the condition between the may and must-transition functions can equivalently be written as $\delta^u_{\mathtt{must}}(s) \subseteq \delta^u_{\mathtt{may}}(s)$.

The predicate valuations $\mathcal{I}_{\mathtt{may}}$ and $\mathcal{I}_{\mathtt{must}}$ can be equivalently represented in the style of Partial Kripke structures [5, 6], by defining a single 3-valued valuation $\mathcal{I}_{\mathtt{3val}} : S \to (\Sigma \to \{\mathtt{T}, \mathtt{F}, \bot\})$, defined for all $s \in S$ and $q \in \Sigma$ by $\mathcal{I}_{\mathtt{3val}}(q)(s) = \mathtt{T}$ if $s \in \mathcal{I}_{\mathtt{must}}(q)$; $\mathcal{I}_{\mathtt{3val}}(q)(s) = \mathtt{F}$ if $s \notin \mathcal{I}_{\mathtt{may}}(q)$; and $\mathcal{I}_{\mathtt{3val}}(q)(s) = \bot$ otherwise.

Definition 2.2 (strategies). Given $A \subseteq \{1, 2\}$, a *may-strategy* (resp. *must-strategy*) π_i for team A is a function $\pi_i : S^+ \mapsto (\mathcal{P}(S) \setminus \emptyset)$ such that, for all $\vec{s} \in S^*$ and all $s \in S$ we have:

- if $\lambda(s) \in A$, then $\pi_i(\vec{s}s) \subseteq \delta_{\mathtt{may}}(s)$ (resp. $\pi_i(\vec{s}s) \in \delta_{\mathtt{must}}(s)$);
- if $\lambda(s) \notin A$, then $\pi_i(\vec{s}s) = S$.

We denote by $\Pi^A_{\mathtt{may}}$ (resp. $\Pi^A_{\mathtt{must}}$) the set of all may (resp. must) strategies for team A. ∎

Two strategies, one for team A and one for team $\neg A$, together with an initial state, determine a set of infinite paths, called *outcomes,* as follows.

Definition 2.3 (outcomes). For all teams $A \subseteq \{1, 2\}$, initial states $s \in S$, modalities $\gamma_A, \gamma_{\neg A} \in \{\mathtt{may}, \mathtt{must}\}$ and strategies $\pi \in \Pi^A_{\gamma_A}$ and $\pi' \in \Pi^{\neg A}_{\gamma_{\neg A}}$, we define the set $\mathrm{Outcomes}(s, \pi, \pi')$ as the set of infinite sequences of states $t_0 t_1 t_2 \cdots$ such that $t_0 = s$ and, for all $k \geq 0$, $t_{k+1} \in \pi(t_0 \cdots t_k) \cap \pi'(t_0 \cdots t_k)$. ∎

2.1 Alternating Refinement

Alternating simulation was introduced independently for turn-based games [1] and general concurrent games [3]. Informally, for the special case of turn-based games, alternating simulation provides a way to specify that the possible moves for a player are increased while those for the other player are decreased. In contrast, the alternating refinement that we define below is *symmetric* between the players, and focuses on the increase in must-transitions and decrease in may-transitions of both players. Thus, if a state s' is refined by s, every must-strategy (for any subset of players) is preserved from s' to s; every may-strategy (for any subset of players) is preserved from s to s'.

Our definition of refinement is a two-player version of the definition of [26]. Intuitively, if a state s refines s', then s' has all the may transitions of s. For must-transitions, on the other hand, we need to take into account the semantics of transitions from a state to a set of states. The definition reflects the view of must transition as games between the system and nondeterminism. Informally, if s refines s', then for every abstract move $U' \in \delta_{\mathtt{must}}(s')$ there is a related system move $U \in \delta_{\mathtt{must}}(s)$, such that for every nondeterministic counter-move $t \in U$, there is $t' \in U'$ with t refining t'. This ensures that, if the system has a winning strategy against nondeterminism in the abstract system, it has such a strategy also in the refined system, ensuring thus the preservation of must strategies.

Definition 2.4 (alternating refinement and bisimulation). Consider two abstract game structures $\mathcal{G} = \langle S, \lambda, \delta_{\mathtt{may}}, \delta_{\mathtt{must}}, \mathcal{I}\rangle$. and $\mathcal{G}' = \langle S', \lambda', \delta'_{\mathtt{may}}, \delta'_{\mathtt{must}}, \mathcal{I}'\rangle$. A relation $R \subseteq S \times S'$ is a refinement if, for all $s \in S$ and $s' \in S'$, we have:

- $s \, R \, s'$ implies that $\mathcal{I}_{\mathtt{3val}}(s) \geq \mathcal{I}'_{\mathtt{3val}}(s')$;
- $\lambda(s) = \lambda(s')$;
- $\forall t \in \delta_{\mathtt{may}}(s) \,.\, \exists t' \in \delta'_{\mathtt{may}}(s') \,.\, t \, R \, t'$;
- $\forall U' \in \delta'_{\mathtt{must}}(s') \,.\, \exists U \in \delta_{\mathtt{must}}(s) \,.\, \forall t \in U \,.\, \exists t' \in U' \,.\, t \, R \, t'$. ∎

Given two game structures $\mathcal{G}$ and $\mathcal{G}'$, there is a maximum refinement preorder that we term[3] $\succeq_{(\mathcal{G},\mathcal{G}')}$. We omit the subscript $(\mathcal{G},\mathcal{G}')$ when clear from the context, and in particular, when $\mathcal{G}$ and $\mathcal{G}'$ are the same game structure.

If $s \succeq_{(\mathcal{G},\mathcal{G}')} s'$ we say that s' *alternatingly abstracts* s or s *alternatingly refines* s'. For finite-branching transition relations, $\succeq_{(\mathcal{G},\mathcal{G}')}$ is the maximum fixed point of a monotone function with closure ordinal ω, ie. $\succeq_{(\mathcal{G},\mathcal{G}')} = \bigcap_j \succeq^j_{(\mathcal{G},\mathcal{G}')}$, where $\succeq^0_{(\mathcal{G},\mathcal{G}')} = S \times S$ and $\succeq^{j+1}_{(\mathcal{G},\mathcal{G}')}$ is defined from $\succeq^j_{(\mathcal{G},\mathcal{G}')}$ using the above definition.

[3] Note that, writing $s \succeq s'$, s is the refinement, and s' the abstraction: the symbol $\succeq$ is used in the opposite direction as it is customary in the discussion of, for instance, trace inclusion.

2.2 Alternating-μ-calculus

Syntax. The formulas of the logic AMC (alternating-time μ-calculus) [2] add next-time operators parameterized by a subset of players to the traditional μ-calculus. Let Σ be a set of predicate symbols, and $\mathcal{V}$ be a set of variables. The set of μ-calculus formulas is generated by the following grammar where $A \subseteq \{1, 2\}$:

$$\phi ::= p \mid x \mid \neg\phi \mid \phi \vee \phi \mid \langle\langle A\rangle\rangle\bigcirc\phi \mid \mu x.\phi,$$

for predicates $p \in \Sigma$ and variables $x \in \mathcal{V}$. In the quantifications $\mu x.\phi$ and its dual $\nu x.\phi$, we require again that all occurrences of x in ϕ have even polarity. $[[A]]\bigcirc\phi$ is defined as the DeMorgan dual of $\langle\langle A\rangle\rangle\bigcirc\phi$. Let a$\mu$calc denote the set of all AMC formulas, and cl-aμcalc the set of all closed AMC formulas.

Semantics. We define two semantics for AMC, the *may* semantics $\mathcal{G}[\![\cdot]\!]_{\mathtt{may}}$ and the *must* semantics $\mathcal{G}[\![\cdot]\!]_{\mathtt{must}}$. Informally, for a closed formula ϕ, a game structure $\mathcal{G}$, and a state s of $\mathcal{G}$, we have $s \in \mathcal{G}[\![\phi]\!]_{\mathtt{must}}$ if ϕ holds at all refinements of s, and $s \in \mathcal{G}[\![\phi]\!]_{\mathtt{may}}$ when there may be a refinement of s at which ϕ holds. In the following, we write simply $[\![\cdot]\!]_{\mathtt{may}}$, $[\![\cdot]\!]_{\mathtt{must}}$ for $\mathcal{G}[\![\cdot]\!]_{\mathtt{may}}$, $\mathcal{G}[\![\cdot]\!]_{\mathtt{must}}$ whenever the game structure is obvious from the context.

Fix a game structure $\mathcal{G} = \langle S, \lambda, \delta_{\mathtt{may}}, \delta_{\mathtt{must}}, \mathcal{I}\rangle$. A *variable environment* is a function $e : \mathcal{V} \mapsto 2^S$ that associates a subset of states to each variable. The basic lookup of propositional information and the monotone boolean operators work as usual:

$$[\![p]\!]^e_\gamma = \mathcal{I}_\gamma(p) \qquad [\![\phi_1 \vee \phi_2]\!]^e_\gamma = [\![\phi_1]\!]^e_\gamma \cup [\![\phi_2]\!]^e_\gamma$$
$$[\![x]\!]^e_\gamma = e(x).$$

Following [20, 27], negation connects the two semantics.

$$[\![\neg\phi]\!]^e_{\mathtt{may}} = S \setminus [\![\phi]\!]^e_{\mathtt{must}} \qquad [\![\neg\phi]\!]^e_{\mathtt{must}} = S \setminus [\![\phi]\!]^e_{\mathtt{may}}$$

The semantics of the fixpoint operator is defined as usual: for $\gamma \in \{\mathtt{may}, \mathtt{must}\}$,

$$[\![\mu x.\phi]\!]^e_\gamma = \bigcap\{T \subseteq S \mid T = [\![\phi]\!]^{e[x:=T]}_\gamma\}$$

We address the strategy quantifiers below.

Recall that in the case of concrete games a state satisfies the formula $\langle\langle A\rangle\rangle\bigcirc\phi$ if the players in A can force ϕ in the next state [2]. In a state of an abstract game structure where a player from A is to make a move, the formula is true in the must-semantics at a state s with $\lambda(s) \in A$ if there is a must-transition to a set of states, all of which make ϕ

true. Informally, this witnessing must-transition cannot be removed in any refinement of s, and thus the A-team will continue to have a strategy to validate ϕ. In a state where a player not from A is to make a move, the formula is true in the must-semantics if all may-transitions lead to states that satisfies ϕ. Informally, any refinement of s has fewer may-transitions, and thus the player not in A will continue to remain unable to invalidate ϕ. The complete semantics for both strategy quantifiers is described below.

$$
\begin{aligned}
& s \in [\![\langle\!\langle A\rangle\!\rangle \bigcirc \phi]\!]^e_{\mathtt{must}} \text{ if} \\
& \quad \exists a \in \delta_{\mathtt{must}}(s) \,.\, a \subseteq [\![\phi]\!]^e_{\mathtt{must}}, \text{ if } \lambda(s) \in A \\
& \quad \forall a \in \delta_{\mathtt{may}}(s) \,.\, a \in [\![\phi]\!]^e_{\mathtt{must}}, \text{ if } \lambda(s) \notin A \\
& s \in [\![\langle\!\langle A\rangle\!\rangle \bigcirc \phi]\!]^e_{\mathtt{may}} \text{ if} \\
& \quad \exists a \in \delta_{\mathtt{may}}(s) \,.\, a \in [\![\phi]\!]^e_{\mathtt{may}}, \text{ if } \lambda(s) \in A \\
& \quad \forall a \in \delta_{\mathtt{must}}(s) \,.\, a \cap [\![\phi]\!]^e_{\mathtt{may}} \neq \emptyset, \text{ if } \lambda(s) \notin A
\end{aligned}
$$

The may and must-semantics are consistent.

Theorem 2.1 (Consistency). *For all AMC-formulas ϕ, for all variable environments e, $[\![\phi]\!]^e_{must} \subseteq [\![\phi]\!]^e_{may}$.*

The consistency theorem enables us to define the 3-valued semantics of AMC, namely $[\![\cdot]\!]_e$, as a function from formulas to $\{\mathtt{T}, \mathtt{F}, \perp\}$ as follows:

$$
[\![\phi]\!]^e_{\mathtt{3val}}(s) = \begin{cases} \mathtt{T}, \text{ if } s \in [\![\phi]\!]^e_{\mathtt{must}} \\ \mathtt{F}, \text{ if } s \notin [\![\phi]\!]^e_{\mathtt{may}} \\ \perp, \text{ otherwise.} \end{cases}
$$

We show that refinement preserves *all* alternating-μ-calculus formulas. In contrast, alternating-simulation only preserves a class of positive formulas of ATL* [3]. The next theorem ensures that when a state s' is refined by a state s, every formula ϕ that is true (resp. false) in s' is also true (resp. false) in s. Thus, in the 3-valued view, the only change that can happen in moving from s' to s is that ϕ evaluates to $\perp$ in s', and either true or false in s. Thus, must-strategies that achieve an alternating μ-calculus goal of both players are preserved from s' to s. Also, if there is a must-strategy for player i that achieves at state s an AMC-goal ϕ in the next state, we are at least guaranteed that there exists a may-strategy for the same player i from s' that achieves the goal ϕ in the next state, when working against any must-strategy of player $\bar{i}$.

Theorem 2.2 (Soundness and Completeness). *Consider a finitely-branching abstract game structure $\mathcal{G}$, and let s, s' be states of $\mathcal{G}$. The following are equivalent.*

1. $s \succeq s'$
2. $\forall \phi \in$ cl-$a\mu$calc : $[\![\phi]\!]_{3val}(s) \geq [\![\phi]\!]_{3val}(s')$
3. $\forall \phi \in$ cl-$a\mu$calc : $s \in [\![\phi]\!]_{may} \Rightarrow s' \in [\![\phi]\!]_{may}$
4. $\forall \phi \in$ cl-$a\mu$calc : $s \in [\![\phi]\!]_{must} \Leftarrow s' \in [\![\phi]\!]_{must}$

The fixpoint free fragment of the logic suffices for completeness, eg. for the implication (2) $\Rightarrow$ (1), it suffices to consider ϕ without fixpoints. The proof of this theorem follows the outline of [25], with extra quantifiers accounting for the conjunctive-must transitions.

We can now make precise our remark on the minimal-element representation of the must-transition function. Given a must-transition function $\delta_{\mathtt{must}} : S \mapsto \mathcal{P}(\mathcal{P}(S))$, we denote by $\downarrow \delta_{\mathtt{must}} : S \mapsto \mathcal{P}(\mathcal{P}(S))$ the transition relation defined by

$$
\downarrow \delta_{\mathtt{must}}(s) = \{T \in \delta_{\mathtt{must}}(s) \mid \forall T' \in \delta_{\mathtt{must}}(s).T' \not\subset T\}.
$$

Theorem 2.3. *For any game structure $\mathcal{G} = \langle S, \lambda, \delta_{may}, \delta_{must}, \mathcal{I}\rangle$, let $\mathcal{G}' = \langle S, \lambda, \delta_{may}, \downarrow \delta_{must}, \mathcal{I}\rangle$. Then, for every $s \in S$, we have $s \preceq_{(\mathcal{G},\mathcal{G}')} s$ and $s \preceq_{(\mathcal{G}',\mathcal{G})} s$. Thus, for all $\phi \in$ cl-$a\mu$calc and $\gamma \in \{\mathtt{may}, \mathtt{must}\}$ we have $\mathcal{G}[\![\phi]\!]_\gamma = \mathcal{G}'[\![\phi]\!]_\gamma$.*

2.3 Games with Linear-Time Objectives

When considering games, it is often natural to consider linear-time objectives, where a player must ensure that all resulting linear sequences of states satisfy given properties [38]. These linear-time properties can be expressed, for instance, by linear-time temporal logic [29] formulas or ω-automata [37]. We will consider here linear-time properties expressed by *parity* acceptance conditions on the state space of the game structure. Any ω-regular property can be specified by a deterministic ω-automaton with a parity accepting condition [37]; by taking the product between this automaton and the game structure, one obtains a game structure with a parity condition defined over the state space.

Given an abstract game structure $\mathcal{G} = \langle S, \lambda, \delta_{\mathtt{may}}, \delta_{\mathtt{must}}, \mathcal{I}\rangle$, consider a tuple $\Delta = \langle T_1, T_2, \ldots, T_m\rangle$ where $T_1, \ldots, T_m$ is a partition of S into disjoint subsets. Given a trace $\sigma = s_0, s_1, s_2, \ldots \in S^\omega$, we denote by $\mathrm{Index}(\sigma, \Delta)$ the largest $i \in \{1, \ldots, m\}$ such that $s_k \in T_i$ for infinitely many $k \in \mathbb{N}$. Then, the *parity* property $\mathrm{parity}(\Delta)$ is defined by $\mathrm{parity}(\Delta) = \{\sigma \in S^\omega \mid \mathrm{Index}(\sigma, \Delta) \text{ is even}\}$. Note that the complement of parity properties is given by $S^\omega \setminus \mathrm{parity}(\langle T_1, \ldots, T_m\rangle) = \mathrm{parity}(\langle \emptyset, T_1, \ldots, T_m\rangle)$; we indicate it by $\neg\,\mathrm{parity}(\langle T_1, \ldots, T_m\rangle)$. We define the winning states with respect to a parity condition as follows.

Definition 2.5 (winning states). The sets $[\langle\!\langle A\rangle\!\rangle \Phi]_{\mathtt{may}}$ and $[\langle\!\langle A\rangle\!\rangle \Phi]_{\mathtt{must}}$ of may and must-winning states for a team $A \subseteq \{1, 2\}$ with respect to a parity property $\Phi \subseteq S^\omega$ are given by:

$$s \in [\langle\!\langle A \rangle\!\rangle \Phi]_{\mathtt{may}} \ \textit{iff} \ \exists \pi \in \Pi^A_{\mathtt{may}} \,.\, \forall \pi' \in \Pi^{\neg A}_{\mathtt{must}} \,.\ \mathrm{Outcomes}(s, \pi, \pi') \subseteq \Phi$$
$$s \in [\langle\!\langle A \rangle\!\rangle \Phi]_{\mathtt{must}} \ \textit{iff} \ \exists \pi \in \Pi^A_{\mathtt{must}} \,.\, \forall \pi' \in \Pi^{\neg A}_{\mathtt{may}} \,.\ \mathrm{Outcomes}(s, \pi, \pi') \subseteq \Phi \quad \blacksquare$$

To express the 3-valued determinacy of games over abstract game structures, let $\neg A = \{1, 2\} \setminus A$ and $\neg\Phi = S^\omega \setminus \Phi$.

Theorem 2.4 (3-valued determinacy). *For all parity conditions $\Phi = \mathrm{parity}(\langle T_1, \dots, T_m \rangle)$ and all $A \subseteq \{1, 2\}$:*

$$[\langle\!\langle A \rangle\!\rangle \Phi]_{must} = S \setminus [\langle\!\langle \neg A \rangle\!\rangle \neg\Phi]_{may}.$$

When team A must-wins for Φ, team $\neg A$ cannot may-win for $\neg\Phi$. Similarly for team $\neg A$ and $\neg\Phi$. Thus, given a linear-time goal Φ and a team A, at each state s exactly one of the following three cases holds:

- team A must-wins for Φ, and team $\neg A$ does not maybe-win for $\neg\Phi$;
- team A maybe-wins for Φ, but does not must-win for Φ, and team $\neg A$ maybe-wins for $\neg\Phi$, but does not must-win for $\neg\Phi$;
- team A cannot may-win for Φ, and team $\neg A$ must-wins for $\neg\Phi$.

To obtain an AMC formula for computing the set of winning states, we assume that there are propositions $q_1, \dots, q_m$ such that, for $1 \le i \le m$, $[\![q_i]\!]_{\mathtt{may}} = [\![q_i]\!]_{\mathtt{must}} = T_i$. Essentially, this means that the condition is specified directly on the state space, as is the case when it is obtained through the product construction with an alternating automaton. Theorem 2.2 ensures that it is possible to evaluate these alternating μ-calculus solution formula on abstractions of the given game structure, obtaining may and must-approximations for the set of winning states.

Given a parity property $\Phi = \mathrm{parity}(\langle T_1, \dots, T_m \rangle)$ and a team $A \subseteq \{1, 2\}$, we indicate by $\phi[\langle q_1, \dots, q_m \rangle, A]$ the alternating μ-calculus formula

$$\phi[\langle q_1, \dots, q_m \rangle, A] = \Upsilon_m X_m . \cdots \Upsilon_1 X_1 . \bigvee_{i=1}^{m} (q_i \wedge \langle\!\langle A \rangle\!\rangle \bigcirc X_i),$$

where Υ_m is ν if m is even, and is μ otherwise. The next result generalizes to 3-valued games a result of [14], and enables the computation of the may and must-winning states.

Theorem 2.5 (3-valued control for linear objectives). *For all parity properties $\Phi = \mathrm{parity}(\langle T_1, \dots, T_m \rangle)$, if there are propositions $q_1, \dots, q_m$ such that, for $1 \le i \le m$, $[\![q_i]\!]_{may} = [\![q_i]\!]_{must} = T_i$, then for all $\gamma \in \{may, must\}$ and for all teams $A \subseteq \{1, 2\}$ we have:*

$$[\langle\!\langle A \rangle\!\rangle \Phi]_\gamma = [\![\phi[\langle q_1, \dots, q_m \rangle, A]]\!]_\gamma .$$

The following theorem states that the above solution formula, when evaluated over abstractions of the game structure on which the goal is defined, still gives rise to 3-valued determinacy. The result follows from by reasoning on the form of $\phi[\Phi, A]$, and from the duality of AMC operators.

Theorem 2.6. *For all $m \ge 0$, propositions $q_1, \dots, q_m$, teams $A \subseteq \{1, 2\}$, and $\gamma \in \{may, must\}$, we have:*

$$[\![\phi[\langle q_1, \dots, q_m \rangle, A]]\!]_\gamma = S \setminus [\![\phi[\langle \mathrm{F}, q_1, \dots, q_m \rangle, \neg A]]\!]_{\overline{\gamma}}$$

where $\overline{may}$ is $must$, $\overline{must}$ is may, and $[\![\mathrm{F}]\!]_{may} = \emptyset$.

3. Abstraction

We construct game abstractions using *player-preserving* abstractions, that maintain the distinction of which player can play at a state. Given a game structure $\mathcal{G} = \langle S, \lambda, \delta_{\mathtt{may}}, \delta_{\mathtt{must}}, \mathcal{I} \rangle$ and a set of states T, we say that a relation $\rho \subseteq S \times T$ is player-preserving for $\mathcal{G}$ if, for all $(s, s'), (t, s') \in \rho$, we have that $\lambda(s) = \lambda(t)$. Given a relation $\rho \subseteq S \times T$ and $U \subseteq S$, $V \subseteq T$, we write $U \circ \rho = \{s' \in T \mid \exists s \in U \,.\, (s, s') \in \rho\}$ and $\rho \circ V = \{s \in S \mid \exists s' \in V \,.\, (s, s') \in \rho\}$. We say that ρ is *total and surjective* if $S \circ \rho = T$ and $S = \rho \circ T$. Given an abstract game structure $\mathcal{G}$ with state space S, and a total and surjective relation $\rho \subseteq S \times T$, we can construct the *abstraction of $\mathcal{G}$ by ρ*, denoted $\mathrm{Abstr}(\mathcal{G}, \rho)$.

Definition 3.1 (Abstraction construction). Consider an abstract game structure $\mathcal{G} = \langle S, \lambda, \delta_{\mathtt{may}}, \delta_{\mathtt{must}}, \mathcal{I} \rangle$, together with a set T and a total and surjective relation $\rho \subseteq S \times T$ that is player-preserving for $\mathcal{G}$. We define $\mathrm{Abstr}(\mathcal{G}, \rho) = \langle T, \lambda', \delta'_{\mathtt{may}}, \delta'_{\mathtt{must}}, \mathcal{I}' \rangle$ as follows. For all $t \in T$ and all $q \in \Sigma$:

$$\lambda'(s') = \lambda(s), \text{ if } s \,\rho\, s'$$
$$\delta'_{\mathtt{may}}(s') = \delta_{\mathtt{may}}(s) \circ \rho$$
$$\delta'_{\mathtt{must}}(s') = \{U' \subseteq T \mid \forall s \in \rho \circ \{s'\} \,.\, \exists U \in \delta_{\mathtt{must}}(s) \,.\ U \circ \rho \subseteq U'\}$$
$$\mathcal{I}'_{\mathtt{may}}(q) = \mathcal{I}_{\mathtt{may}}(q) \circ \rho$$
$$\mathcal{I}'_{\mathtt{must}}(q) = \{s' \in T \mid \rho \circ \{s'\} \subseteq \mathcal{I}_{\mathtt{must}}(q)\}. \quad \blacksquare$$

Abstraction construction for transition structures can be defined simply by omitting the definition of λ'. In the spirit of the definitions of [36], the abstractions for the states of each player are performed separately following our treatment of abstract transition systems. For each player, the two components of the definition follow [10, 11].

The following theorem serves as a "sanity check" and ensures that Definition 3.1 yields abstractions that are sound w.r.t. refinement.

Theorem 3.1. *Consider an abstract game structure $\mathcal{G} = \langle S, \lambda, \delta_{may}, \delta_{must}, \mathcal{I}\rangle$. Let T be a set of states and let $\rho \subseteq S \times T$ be a total and surjective relation that is player-preserving for $\mathcal{G}$. Let $\mathcal{G}' = \mathrm{Abstr}(\mathcal{G}, \rho)$. Then, for all $s\rho t$, we have $s \succeq_{(\mathcal{G},\mathcal{G}')} t$.*

Abstract interpretation is a rich source of examples for the relations required to use the schema presented by the above definition. The following example, based on [16], illustrates how predicate abstraction fits into this framework.

Example 3.1 (Predicate abstraction). Let $\mathcal{R}$ be a concrete transition structure, whose infinite state space S is given by all possible valuations of three integer variables $\mathtt{x}$, $\mathtt{y}$, and $\mathtt{z}$. Any state c is of the form $\{\mathtt{x} \mapsto i, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$, for some integers i, j, k. Let us assume that the transitions are those induced by the single assignment statement $\mathtt{x} = \mathtt{z}$, e.g. there is a transition from state c above to state $c' = \{\mathtt{x} \mapsto k, \mathtt{y} \mapsto j, \mathtt{z} \mapsto k\}$. The predicates $\phi_1 = \mathtt{odd}(\mathtt{x}), \phi_2 = (\mathtt{y} > 0)$, and $\phi_3 = (\mathtt{z} < 0)$ induce an equivalence relation on the states of $\mathcal{A}_1$: two states are equivalent if they agree on all three predicates. Let T be the set of all equivalence classes of states of $\mathcal{R}$. Define $\rho \subseteq S \times \mathcal{P}(S)$ as the relation associating each state of S to the equivalence class to which it belongs. By definition of abstraction, there is a may-transition from a to a' in $\mathrm{Abstr}(\mathcal{R}, \rho)$ iff there are $c \in a$ and $c' \in a'$ such that c has a transition to c' in $\mathcal{R}$. Dually, there is a must-transition from a to A iff, *for all* $c \in a$, there exists $a' \in A$ and $c' \in a'$ such that c has a transition to c' in $\mathcal{R}$. For instance, there is a may-transition from the state $\phi_1 \wedge \phi_2 \wedge \phi_3$ to each of the states $\phi_1 \wedge \phi_2 \wedge \phi_3$ and $\neg\phi_1 \wedge \phi_2 \wedge \phi_3$. There is a $\mathtt{must}$-transition from $\phi_1 \wedge \phi_2 \wedge \phi_3$ to *the set* $\{(\phi_1 \wedge \phi_2 \wedge \phi_3, \neg\phi_1 \wedge \phi_2 \wedge \phi_3\}$ that captures the "absence of effect" of the statement on $\mathtt{y}$ and $\mathtt{z}$. ∎

The must-transition of the above example, which leads to a set of states, was missing in our earlier treatment of abstraction via MTSs [16]. We showed in that paper that the absence of this transition causes the framework to be unable to support incremental abstractions. In this paper, we do have the required compositionality to ensure that abstractions can be built incrementally.

Theorem 3.2 (Compositionality of abstractions). *Let $\mathcal{G} = \langle S, \lambda, \delta_{may}, \delta_{must}, \mathcal{I}\rangle$ be an* abstract game structure. *Let T, U be sets of states and $\rho \subseteq S \times T$ and $\rho' \subseteq T \times U$ be total and surjective, such that both ρ and $\rho' \circ \rho$ are player-preserving for $\mathcal{G}$. Then,*

$$\mathrm{Abstr}(\mathcal{G}, \rho' \circ \rho) = \mathrm{Abstr}(\mathrm{Abstr}(\mathcal{G}, \rho), \rho')$$

An equivalent result for transition structures can be obtained simply by omitting the player-preserving requirement from the above definition.

3.1 Precision of abstraction

We now explore the precision of the abstraction defined in definition 3.1. For notational simplicity, our results are phrased in terms of abstract transition structures, but analogous results hold for abstract game structures.

The basic algorithmic problem in the evaluation of μ-calculus formulas consists in computing the predecessor operators $\exists\,\mathrm{pre}_\gamma$. must be possible to compute in the the operators system. Specifically, we show that once the abstract state space and the abstraction relation ρ are chosen, the three-valued abstractions proposed in this paper, with conjunctive representation for must-transitions, enable the computation of the predecessor operators on the abstract system in a way that is "as precise as possible", for the chosen abstract state space and abstraction relation.

We consider a concrete abstract transition structure $\mathcal{R} = \langle S, \delta_{\mathtt{may}}, \delta_{\mathtt{must}}, \mathcal{I}\rangle$, together with a set T, and a surjective and total relation $\rho \subseteq S \times T$. The idea underlying the construction of abstractions via ρ is that, when the abstract structure is at a state $t \in T$, the concrete structure can be at any state in $\rho \circ \{t\}$. To take into account this uncertainty, given a set $U \subseteq T$ and $\gamma \in \{\mathtt{must}, \mathtt{may}\}$, we relate the computation of $\exists\,\mathrm{pre}_\gamma(U)$ to the following game, played from a state $t \in T$. The game involves two players: Proponent, that tries to reach U from t in one step (as called for by $\exists\,\mathrm{pre}_\gamma(U)$), and Spoiler, that tries to prevent this. Spoiler plays first, and chooses a state $s \in \rho \circ \{t\}$ in the concrete structure that is related to t. Then, Proponent (if $\gamma = \mathtt{may}$) or Spoiler (if $\gamma = \mathtt{must}$) chooses a move $a \in \delta_\gamma(s)$. If $\gamma = \mathtt{must}$, Spoiler chooses $s' \in a$; otherwise, if $\gamma = \mathtt{may}$, Spoiler chooses $s' = a$. Finally, Spoiler chooses a state $t' \in \{s\} \circ \rho$. Thus, Spoiler represents both the imprecision involved in moving between the concrete system and the abstraction, and the conjunctive nature of the must-transitions.

We say that an abstraction via ρ of $\mathcal{R}$ is *precise* if for all $\gamma \in \{\mathtt{must}, \mathtt{may}\}$ and $U \subseteq S$, we can compute $\exists\,\mathrm{pre}_\gamma(U)$ on the abstraction such that for $t \in T$, we have $t \in \exists\,\mathrm{pre}_\gamma(U)$ iff Proponent has a strategy to reach U from t in the above game. For a formalization of precision using the language of abstract interpretation see [10, 11] — the notable difference in our results is the difference in the types of may and must transition relations.

Theorem 3.3 (Precision of abstraction). *Given an abstract transition structure $\mathcal{R}$ with state space S and a total and surjective function $\rho \subseteq S \times T$ for some T, the abstraction $\mathrm{Abstr}(\mathcal{R}, \rho)$ is a precise abstraction of $\mathcal{R}$ via ρ.*

The above theorem differs from the results of [8] on optimal abstractions. In [8], the structure of the abstraction is fixed (both may and must-transitions have type $S \mapsto \mathcal{P}(S)$, and are thus non-conjunctive); *among abstractions having that structure,* it is proved that the abstraction constructed there is as precise as possible. In contrast, in our result the structure of the abstraction is unconstrained — the abstraction of $\mathcal{R}$ by ρ constructed by $\mathrm{Abstr}(\mathcal{R}, \rho)$ is the *most precise possible* of all abstractions of $\mathcal{R}$ by ρ. Similar results hold for the abstraction of abstract game structures.

3.2 Completeness of abstraction

Consider AMC with only greatest fixed points.

$$\begin{aligned}\phi ::= {} & p \mid \neg p \mid x \mid \phi \vee \phi \mid \phi \wedge \phi \\ & \mid \langle\langle A \rangle\rangle \bigcirc \phi \;\mid [[A]] \bigcirc \phi \mid \nu x.\phi\end{aligned}$$

This is a logic of safety properties. In the special case of abstract transition structures, the logic includes both universal-safety and existential-safety properties in the terminology of [30].

The following theorem shows that, for any possibly-infinite concrete game structure $\mathcal{G}$ and any formula ϕ of this logic, there exists a *finite* abstraction of this game structure that preserves the truth value (T or F) of ϕ interpreted on $\mathcal{G}$.

Theorem 3.4. *Let a concrete game structure $\mathcal{G} = \langle S, \lambda, \delta, \mathcal{I} \rangle$, $s \in S$ and a formula ϕ in above logic be such that $s \in [\![\phi]\!]_{must}$. Then, there is a* finite *set T and a total and surjective $\rho \subseteq S \times T$ yielding a finite abstract game structure $\mathrm{Abstr}(\mathcal{G}, \rho)$ satisfying :*

$$(\forall a \in A)\ [s\ \rho\ a \Rightarrow\ a \in [\![\phi]\!]_{must}]$$

The abstraction relation ρ required for the proof of the above theorem is obtained from an equivalence relation that relates concrete states if they are not distinguished by any subformulas of ϕ.

This completeness result is clearly "foretold" in the completeness results of [32]; indeed, elsewhere in this volume [12] uses the techniques of [32] to show that the addition of fairness constraints to abstract transition structures permits the above theorem to be proved for arbitrary μ-calculus formulas.

4 Model Checking and Refinement Checking

In this section, we discuss the complexity of the model checking[4] and refinement checking problems for abstract transition and game structures via a reduction from abstract game structures to the traditional (concrete, turn-based) game structures of [2].

[4]Symbolic model-checking algorithms can be obtained directly from the semantics of the μ-calculus on abstract transition structures in Section 2 and of the alternating μ-calculus on abstract game structures in Section 3.

Given an abstract game structure $\mathcal{G} = \langle S, \lambda, \delta_{\mathtt{may}}, \delta_{\mathtt{must}}, \mathcal{I} \rangle$ over a set of propositions Σ, we define a concrete 2-valued game structure $\mathcal{G}' = \langle S', \lambda', \delta', \mathcal{I}' \rangle$ over a set of propositions $\Sigma' = \Sigma \cup \{\bar{p} \mid p \in \Sigma\} \cup \{p_{\mathtt{must}}, p_1, p_2\}$ as follows.

- $S' = S \cup \{\langle T, i \rangle \mid \exists s \in S : T \in \delta_{\mathtt{must}}(s) \wedge i = \overline{\lambda(s)}\}$.
- $\lambda' : S' \mapsto \{1, 2\}$ is defined as $\lambda'(s) = \lambda(s)$ for $s \in S$ and as $\lambda'(\langle T, i \rangle) = i$ for $T \in \mathcal{P}(S)$.
- The transition function $\delta' : S' \mapsto S'$ is defined, for all $s \in S$, by:

  $$\delta'(s) = \delta_{\mathtt{may}}(s) \cup \{\langle T, i \rangle \mid T \in \delta_{\mathtt{must}}(s) \wedge i = \overline{\lambda(s)}\}$$

  and for all states of the form $\langle T, i \rangle$ in S', by $\delta'(\langle T, i \rangle) = \{s' \mid s' \in T\}$.
- The interpretation function $\mathcal{I}' : S \to [\Sigma' \to \{\mathrm{T}, \mathrm{F}\}]$ is defined, for $s \in S$, $p \in \Sigma'$, $i \in \{1, 2\}$ by:

  $$\begin{array}{ll}\mathcal{I}'(s)(p) = (\mathcal{I}(s)(p) \neq \mathrm{F}) & \mathcal{I}'(s)(p_i) = (\lambda(s) = i) \\ \mathcal{I}'(s)(\bar{p}) = (\mathcal{I}(s)(p) \neq \mathrm{T}) & \mathcal{I}'(s)(p_{\mathtt{must}}) = \mathrm{F}.\end{array}$$

  For all states of the form $\langle T, i \rangle$ in S', and for all $p \in \Sigma'$, we have $\mathcal{I}'(\langle T, i \rangle)(p_{\mathtt{must}}) = \mathrm{T}$.

Intuitively, the traditional game structure $\mathcal{G}'$ encodes each set T of successor states in $\mathcal{G}$ defined in conjunctive must-transitions by a state of the form $\langle T, i \rangle$ from which all states s' in T are then reachable. Also, the interpretation function $\mathcal{I}'$ for propositions p and $\bar{p}$ is designed to preserve the may-semantics $[\![\cdot]\!]_{\mathtt{may}}$. Since the size of $\mathcal{G}$ is defined as $|\mathcal{G}| = \sum_{s \in S}(|\delta_{\mathtt{may}}(s)| + \sum_{a \in \delta_{\mathtt{must}}(s)} |a|)$, we have $|\mathcal{G}'| = O(|\mathcal{G}|)$.

Given an AMC formula ϕ, we define a recursive formula transformation $T(\phi)$ as follows. First, we rewrite ϕ in positive normal form, pushing negation inwards using De-Morgan's laws, and then we apply recursively the following rewriting rules: $T(p) = p$, $T(\neg p) = \bar{p}$, $T(\phi_1 \vee\ \phi_2) = T(\phi_1) \vee T(\phi_2)$, $T(\phi_1 \wedge\ \phi_2) = T(\phi_1) \wedge T(\phi_2)$, $T(x) = x$, $T(\mu x.\phi) = \mu x.T(\phi)$, $T(\nu x.\phi) = \nu x.T(\phi)$, and

$$\begin{aligned}T(\langle\langle A \rangle\rangle \bigcirc \phi) = {} & \Big[\bigvee_{i \in A} p_i \vee\ \langle\langle A \rangle\rangle \bigcirc (\neg p_{\mathtt{must}} \vee \langle\langle A \rangle\rangle \bigcirc T(\phi))\Big] \\ & \wedge \Big[\neg(\bigvee_{i \in A} p_i) \vee\ \langle\langle A \rangle\rangle \bigcirc (\neg p_{\mathtt{must}} \wedge T(\phi))\Big] \\ T([[A]] \bigcirc \phi) = {} & \Big[\neg(\bigvee_{i \in A} p_i) \vee [[A]] \bigcirc (\neg p_{\mathtt{must}} \vee [[A]] \bigcirc T(\phi))\Big] \\ & \wedge \Big[\bigvee_{i \in A} p_i \vee\ [[A]] \bigcirc (\neg p_{\mathtt{must}} \wedge T(\phi))\Big].\end{aligned}$$

The correctness of these game and formula translations is defined by showing that, for all formulas ϕ and states s in S, we have $[\![\phi]\!]_{\mathtt{may}}(s) = \mathrm{F}$ in $\mathcal{G}$ iff $[\![T(\phi)]\!](s') = \mathrm{F}$ in $\mathcal{G}'$, where

s' is the state corresponding to s in $\mathcal{G}'$. This result gives us a decision procedure for solving the 3-valued model-checking problem on abstract game structures (which generalizes the 3-valued model-checking procedure of [6, 16] for closed systems):

- if $[\![T(\phi)]\!](s') = \mathtt{F}$ in $\mathcal{G}'$, then $[\![\phi]\!]_{\mathtt{3val}}(s) = \mathtt{F}$ in $\mathcal{G}$;
- if $[\![T(\neg\phi)]\!](s') = \mathtt{F}$ in $\mathcal{G}'$, then $[\![\phi]\!]_{\mathtt{3val}}(s) = \mathtt{T}$ in $\mathcal{G}$;
- otherwise, $[\![\phi]\!]_{\mathtt{3val}}(s) = \perp$ in $\mathcal{G}$.

From this construction, we obtain the following result.

Theorem 4.1. *The AMC model-checking problem for abstract game structures and the AMC model-checking problem for concrete game structures are inter-reducible in linear time and logarithmic space.*

Thus, both lower and upper complexity bounds for AMC model checking over concrete games carry over to abstract games. The game and formula translations described above can also be used to prove that the same results hold for other 3-valued alternating temporal logics whose semantics is defined by following the same rules as those used in Section 3 to define the 3-valued AMC from the traditional AMC. See [2] for the complexity of model checking for various alternating temporal logics.

The construction of $\mathcal{G}'$ and $T(\phi)$ described above is also applicable to abstract transition structures, i.e., abstract game structures with only one player. In that case, all the moves are played by the system, except in states of the form $\langle T, i\rangle$ where the environment picks the next state. Thus, model checking on abstract transition structures is also reducible (in linear time and logarithmic space) to two model-checking problems on traditional game structures. Conversely, it can be shown that the game played by two players of a traditional game structure can be simulated by the game played by the system and its environment in an abstract game structure.

Theorem 4.2. *The AMC model-checking problem for abstract transition structures and the AMC model-checking problem for concrete game structures are inter-reducible in linear time and logarithmic space.*

Thus, the alternation expressible in abstract transition structures makes them compact and precise, but increases the cost of model checking compared to traditional MTSs, LTSs or Kripke structures. In the case of CTL/ATL, the cost in the size of the structure simply increases from NLOGSPACE-complete for CTL to PTIME-complete for ATL, while both CTL and ATL model checking can be done in linear time [2]. However, in the case of CTL*/ATL*, the cost of model checking increases from PSPACE-complete for CTL* to 2EXPTIME-complete for ATL* [2].

Checking alternating refinement between abstract transition or game structures can also be reduced to checking alternating simulation on traditional game structures obtained using a construction similar to the one of $\mathcal{G}'$ above. Since checking alternating simulation can be done in PTIME [3], and since checking alternating refinement generalizes checking simulation which is itself PTIME-hard (see [3]), checking alternating refinement is PTIME-complete.

Theorem 4.3. *Given two abstract game structures $\mathcal{G}$ and $\mathcal{G}'$, checking whether $\mathcal{G} \succeq \mathcal{G}'$ is PTIME-complete.*

Acknowledgements

We thank Kedar Namjoshi and the anonymous reviewers for their helpful comments. This work was supported in part by the NSF grants CCR-0132780, CCR-0234690, CCR-0244901, CCR-0341658, and by the ONR grant N00014-02-1-0671.

References

[1] S. Abramsky. Semantics of interaction. In *Trees in Algebra and Programming – CAAP'96, Proc. 21st Int. Coll., Linköping*, volume 1059. Springer-Verlag, 1996.

[2] R. Alur, T. Henzinger, and O. Kupferman. Alternating time temporal logic. *J. ACM*, 49:672–713, 2002.

[3] R. Alur, T. Henzinger, O. Kupferman, and M. Vardi. Alternating refinement relations. In *CONCUR 98: Concurrency Theory. 9th Int. Conf.*, volume 1466 of *Lect. Notes in Comp. Sci.*, pages 163–178. Springer-Verlag, 1998.

[4] T. Ball, A. Podelski, and S. K. Rajamani. Boolean and Cartesian Abstraction for Model Checking C Programs. In *Proceedings of TACAS'2001*, 2001.

[5] G. Bruns and P. Godefroid. Model Checking Partial State Spaces with 3-Valued Temporal Logics. In *Proceedings of the 11th Conference on Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 274–287. Springer Verlag, July 1999.

[6] G. Bruns and P. Godefroid. Generalized Model Checking: Reasoning about Partial State Spaces. In *Proceedings of CONCUR'2000 (11th International Conference on Concurrency Theory)*, volume 1877 of *Lecture Notes in Computer Science*, pages 168–182. Springer Verlag, August 2000.

[7] E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. In *Proc. 19th ACM Symp. Princ. of Prog. Lang.*, pages 343–354, 1992.

[8] R. Cleaveland, P. Iyer, and D. Yankelevich. Optimality in abstractions of model checking. In *SAS'95: Proc. 2d. Static Analysis Symposium*, Lecture Notes in Computer Science 983, pages 51–63. Springer, 1995.

[9] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Pasareanu, Robby, and H. Zheng. Bandera: Extracting Finite-state Models from Java Source Code. In *Proceedings of the 22nd Intl' Conference on Software Engineering*, June 2000.

[10] D. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. PhD thesis, Technical University of Eindhoven, 1996.

[11] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, 1997.

[12] D. Dams and K. Namjoshi. The existence of finite abstractions for branching time model checking. In *Proceedings of the 19th IEEE conference on Logic in Computer Science*. IEEE, 2004. These proceedings.

[13] L. de Alfaro, T. Henzinger, and F. Mang. Detecting errors before reaching them. In *CAV 00: Proc. of 12th Conf. on Computer Aided Verification*, volume 1855 of *Lect. Notes in Comp. Sci.*, pages 186–201. Springer-Verlag, 2000.

[14] E. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proc. 32nd IEEE Symp. Found. of Comp. Sci.*, pages 368–377. IEEE Computer Society Press, 1991.

[15] P. Godefroid. Reasoning about Abstract Open Systems with Generalized Module Checking. In *Proceedings of EMSOFT'2003 (3rd Conference on Embedded Software)*, volume 2855 of *Lecture Notes in Computer Science*, pages 223–240, Philadelphia, October 2003. Springer-Verlag.

[16] P. Godefroid, M. Huth, and R. Jagadeesan. Abstraction-based Model Checking using Modal Transition Systems. In *Proceedings of CONCUR'2001 (12th International Conference on Concurrency Theory)*, volume 2154 of *Lecture Notes in Computer Science*, pages 426–440, Aalborg, August 2001. Springer-Verlag.

[17] T. Henzinger, R. Jhala, R. Majumdar, G. Necula, G. Sutre, and W. Weimer. Temporal-safety proofs for systems code. In *CAV 02: Proc. of 14th Conf. on Computer Aided Verification*, volume 2404 of *Lect. Notes in Comp. Sci.*, pages 526–538. Springer-Verlag, 2002.

[18] T. Henzinger, R. Majumdar, F. Mang, and J.-F. Raskin. Abstract interpretation of game properties. In *SAS 00: Static Analysis*, Lecture Notes in Computer Science 1824, pages 220–239. Springer-Verlag, 2000.

[19] M. Huth, R. Jagadeesan, and D. Schmidt. Modal transition systems: a foundation for three-valued program analysis. In *Proceedings of the European Symposium on Programming (ESOP'2001)*, volume 2028. Springer Verlag, April 2001.

[20] P. Kelb. Model checking and abstraction: a framework preserving both truth and failure information. Technical Report OFFIS, University of Oldenburg, Germany, 1994.

[21] S. Kremer and J.-F. Raskin. A game-based verification of non-repudiation and fair exchange protocols. In *Proceedings of the 12th International Conference on Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*, pages 551–565. Springer, 2001.

[22] S. Kremer and J.-F. Raskin. Game analysis of abuse-free contract signing. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 206–220, 2002.

[23] O. Kupferman and M. Vardi. Module Checking. In *Proc. 8th Conference on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 75–86, New Brunswick, August 1996. Springer-Verlag.

[24] K. G. Larsen. Modal Specifications. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, number 407 in Lecture Notes in Computer Science, pages 232–246. Springer Verlag, June 12–14 1989. International Workshop, Grenoble, France.

[25] K. G. Larsen and B. Thomsen. A Modal Process Logic. In *Third Annual Symposium on Logic in Computer Science*, pages 203–210. IEEE Computer Society Press, 1988.

[26] K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *Proceedings of the 5th IEEE conference on Logic in Computer Science*, pages 108–117. IEEE, 1990.

[27] F. Levi. A symbolic semantics for abstract model checking. In *Static Analysis Symposium: SAS'98*, volume 1503 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

[28] 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: An International Journal*, 6(1):11–44, January 1995.

[29] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.

[30] P. Manolios and R. J. Trefler. Safety and liveness in branching time. In *Proceedings of the 16th IEEE conference on Logic in Computer Science*, pages 366–375. IEEE, 2001.

[31] R. Milner. An algebraic definition of simulation between programs. In *Second International Joint Conference on Artificial Intelligence*, pages 481–489. The British Computer Society, 1971.

[32] K. S. Namjoshi. Abstraction for branching time properties. In *Proceedings of the 15th International Conference on Computer Aided Verification*, volume 2725 of *Lecture Notes in Computer Science*, pages 288–300. Springer, 2003.

[33] C. S. Pasareanu, M. B. Dwyer, and W. Visser. Finding feasible counter-examples when model checking abstracted Java programs. *Lecture Notes in Computer Science*, 2031, 2001.

[34] S. Shoham. A game-based framework for ctl counter-examples and 3-valued abstraction-refinement. Master's thesis, TECHNION - Israel Institute of Technology, 2003. Paper with O. Grumberg in Proceedings of CAV 2003.

[35] S. Shoham and O. Grumberg. Monotonic abstraction-refinement for ctl. In *Tools and Algorithms for the Construction and Analysis of Systems: 10th International Conference*, number 2988 in Lecture Notes in Computer Science, pages 546–560. Springer Verlag, 2004.

[36] P. Stevens. Abstract games for infinite state processes. In *Proceedings of the 9th International Conference on Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, 1998.

[37] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 135–191. Elsevier Science Publishers (North-Holland), Amsterdam, 1990.

[38] W. Thomas. On the synthesis of strategies in infinite games. In *Proc. of 12th Annual Symp. on Theor. Asp. of Comp. Sci.*, volume 900 of *Lect. Notes in Comp. Sci.*, pages 1–13. Springer-Verlag, 1995.